Resilient and energy-efficient scheduling algorithms at scale

Guillaume Aupy

N° attribué par la bibliothèque: 2014ENSL0928

- ÉCOLE NORMALE SUPÉRIEURE DE LYON -
Laboratoire de l'Informatique du Parallélisme - UMR5668 - LIP

THÈSE

en vue d'obtenir le grade de

Docteur de l'Université de Lyon, délivré par l'École Normale Supérieure de Lyon

Spécialité : Informatique

au titre de l'École Doctorale Informatique et Mathématiques

présentée et soutenue publiquement le 16 Septembre 2014 par

Guillaume AUPY

Resilient and energy-efficient scheduling algorithms at scale

Directrice de thèse :	Anne	BENOÎT
Co-encadrant de thèse :	Yves	ROBERT

Devant la commission d'examen formée de :

Anne	BENOÎT	*Directrice*
Georges	DA COSTA	*Examinateur*
Erol	GELENBE	*Examinateur*
Alain	GIRAULT	*Rapporteur*
Alix	MUNIER KORDON	*Rapporteuse*
Yves	ROBERT	*Co-encadrant*

Remerciements

Je voudrais d'abord remercier mes deux rapporteurs, Alain Girault et Alix Munier, d'avoir accepté de relire en détail ce manuscrit. Je me rends compte du temps que ça a pu leur prendre, et j'imagine qu'il peut y avoir lecture plus agréable en juillet sur la plage ou à la montagne. Je suis également très reconnaissant à Georges Da Costa d'avoir pris part à mon jury de soutenance, à Erol Gelenbe de l'avoir présidé et pour les intéressantes discussions qui ont suivies. Et même si il ne fait pas partie du jury, je remercie énormément Bora qui, avant que je n'envoie ma thèse à mes rapporteurs, a proposé de relire le manuscrit et en a corrigé quelques centaines de fautes !

Anne, Yves, il est assez clair que sans vous je n'en serais pas aussi loin, merci beaucoup pour votre aide, direction (parfois contre moi même ☺) et surtout de m'avoir supporté pendant un peu plus de trois années. Ce n'est pas toujours facile !

Pendant ces trois années, j'ai eu la chance d'avoir pu travailler avec énormément de monde, collaborations qui se sont toujours extrèmement bien passées, merci à Ana, Dounia, Fanny, Franck, Frédo, Henri, Jakub, Manu, Mathieu, Matthieu, Padma, Paul, Piotr, Rami, Thomas. Ces collaborations m'ont apporté énormément par différentes visions, nouveaux outils et manières de travailler.

Le LIP est un endroit génial pour faire une thèse. Je remercie énormément les différent-e-s assistant-e-s d'équipe : Évelyne et Lætitia qui ont géré mes missions et ont répondu à toutes mes questions pendant ces trois années, Damien qui avait la dure charge de nous simplifier les procédures d'inscription et de réinscription en thèse, mais aussi Catherine, Chiraz, Marie, Séverine et Sylvie, qui m'ont toutes aussi, à un moment donné, donnée un coup de main. Enfin, merci à Dom et Serge pour le soutien IT.

Yves m'a souvent dit pendant la thèse qu'il ne comprenait pas comment j'avais le temps de travailler avec tout ce que je faisais à côté, je pense au contraire que c'est grâce à tous les gens qui m'aidaient à me changer les idées que j'ai pu aboutir à ce manuscrit. Merci donc à mes partenaires et adversaires de babyfoot, de coinche, aux joueurs de foot du mercredi midi, à mon prof de surf. Merci aussi aux buveurs de bières, de chartreuse, à la liste impega, aux parents de Paul qui nous ont prêté leur chalet pour les WE croziflettes. Merci à l'équipe de Maths en Jeans, de la MMI et de Plaisir-Maths.

Enfin, je remercie Laura, et pas uniquement parce qu'elle nous a "fait découvrir l'incroyable similitude entre notre formule d'énergie et $E = mc^2$" (Paul R-G, 2012), mais parce que, en général.

Enfin, merci à tou-te-s celles et ceux qui ont pris le temps de venir voir ma soutenance, à celles et ceux qui m'ont aidé à organiser le pot. Et merci au lecteur qui a au moins lu ça (voire plus ?). Pour sa peine, l'Algorithme 0 présente la recette de la croziflette au canard dont la conception nous a demandé beaucoup de week-end éprouvants à la montagne et est, à sa manière, aussi un aboutissement de ma thèse.

Thanks

I would like to thank Alain Girault and Alix Munier, my two reviewers, for their work, Georges Da Costa and Erol Gelenbe for joining my thesis committee. I thank all of them for attending my defense. I further would like to thank all the people I got to work with during my PhD: Ana, Dounia, Fanny, Franck, Frédo, Henri, Jakub, Manu, Mathieu, Matthieu, Padma, Paul, Piotr, Rami, Thomas.

Algorithm 0: Croziflette au canard (pour 4 personnes)

ingrédients: 4 cuisses de confit de canard
200g de crozets au sarrasin
200g de crozets au froment
1 reblochon (ou plus)
2 bouteilles de bordeaux

`/* Avec les crozets qui restent (paquets de 400g), ne pas hésiter à faire sa petite sœur où l'on remplace le canard par du saumon fumé et de la crème fraiche. */`

begin

for {*crozets, canard, reblochon* } **do in parallel**

Crozets Faire cuire les crozets ensemble

Canard

1. Nettoyer le gras autour des cuisses de canard (*tout* le gras)
2. Émietter le canard dans le plat à croziflette
3. Manger ce qui reste autour des os

Reblochon

1. Couper le reblochon en 3 dans le sens de l'épaisseur.
2. Couper la tranche du milieu en cubes
3. Ne pas manger les cubes

Verser les crozets cuits dans le plat;
Mélanger ;
Insérer les cubes de reblochons dans le mélange;
Paver la face supérieure du mélange avec les deux tranches de reblochons restantes, *croûte en dessous*;
Faire cuire à 200°C jusqu'à ce que le reblochon du dessus ait une jolie couleur;
return *Croziflette, bouteilles de bordeaux*

Contents

Résumé en français

Dans cette thèse, nous avons considéré d'un point de vue théorique deux problèmes importants pour les futures plateformes dites "exascales" (plateformes pouvant effectuer 10^{18} opérations par secondes) : les restrictions liées à la fiabilité de ces plateformes ainsi que les contraintes énergétiques.

Les contraintes de fiabilité étaient connues avant même l'ère "petascale" (plateformes pouvant effectuer 10^{15} opérations par secondes). Alors que la fiabilité des composants pris de manière indépendante augmente, leur nombre aussi augmente de manière exponentielle. Le temps moyen entre deux fautes (MTBF) des machines hautes performances est proportionnel au MTBF de ses composants, mais aussi à l'inverse du nombre de processeurs sur ces machines. Ce MTBF décroit donc rapidement. Pour les machines "exascale", il est attendu que ce temps moyen soit plus faible que le temps pour faire une sauvegarde ("checkpoint") des données présentes sur la machine. En première partie de cette thèse, nous nous sommes intéressés à l'étude de placement optimal de ces checkpoints dans un but de minimisation du temps total d'exécution. En particulier, nous avons considéré les checkpoints périodiques (temps de travail, en l'absence de fautes, constant entre deux checkpoints), et coordonnés (la machine sauve de manière simultanée les données de tous les processeurs qui travaillent). Nous avons commencé par un contexte général et nous avons amélioré la période, optimale au premier ordre, donnée par Young et Daly. Puis, dans un contexte exascale, nous avons considéré des prédicteurs de fautes. Un prédicteur de fautes est un logiciel lié à la machine étudiée, qui par l'étude de "logs" (les événements passés) et d'informations données par des capteurs sur la machine, va tenter de prédire lorsqu'une faute va arriver. Ce prédicteur est évidemment imparfait, on suppose qu'il ne pourra prédire qu'un certain pourcentage r, appelé "recall", de ces fautes, et qu'en plus parmi ses prédictions, seul un certain pourcentage p, appelé "precision", seront effectivement des futures fautes (les autres prédictions étant ce qu'on appelle des faux-positifs). Évidemment, seules les fautes prédites suffisamment de temps en avance pour d'éventuelles actions pro-actives sont comptabilisées. En effet, un prédicteur qui prédirait parfaitement toutes les fautes, mais au moment où elles arrivent ne servirait à rien ! Enfin, ces prédicteurs ne peuvent évidemment pas prévoir exactement le moment où la faute va arriver, mais peuvent donner un "intervalle de confiance", I, dans lequel ils s'attendent à ce que leur faute arrive. Dans ce contexte, nous avons proposé des algorithmes efficaces, et pour ces algorithmes, donné des formules optimales au premier ordre pour le temps entre deux checkpoints. Une hypothèse importante de ce travail est que l'utilisateur connaît exactement le moment où la faute arrive : la machine s'arrête, un processeur surchauffe, le travail courant renvoie une erreur. Dans un deuxième temps, nous avons considéré les fautes dites silencieuses. Ces fautes, provoquées par exemple par un rayon cosmique entraînant l'inversion d'un bit, sont de plus en plus courantes, et ne peuvent être détectées que par un système de vérification. Dans le cas où l'une de ces fautes est détectée, l'utilisateur doit retourner au point de sauvegarde le plus récent qui n'a pas été affecté par cette faute, si un tel point existe ! Dans le cas de ces fautes, il faut donc garder en mémoire plusieurs points de sauvegarde au cas où l'un ou plusieurs d'entre eux aient été affectés par une telle faute. Dans ce contexte, nous avons à nouveau proposé des algorithmes optimaux au premier ordre, mixant points de sauvegarde et points de vérification.

Dans la seconde partie de cette thèse, nous avons considéré des problèmes énergétiques liés à ces mêmes plateformes. En 2004, la consommation énergétique des processeurs a atteint un seuil, tel qu'un dépassement de ce seuil faisait craindre la fonte de ces processeurs (à cause des dégagements de chaleur liés à la consommation énergétique). Ce jour là, la course à la puissance de calcul de ces processeurs s'est arrêtée (ainsi que la fameuse "Loi de Moore", loi énoncée par Moore, qui prévoyait un doublement de la puissance de calcul des processeurs tous les 18 mois). Il faut donc trouver de nouveaux paradigmes pour diminuer cette consommation énergétique, sans quoi l'exascale risque de ne jamais voir le jour (pour rappel, une machine exascale doit avoir une puissance de calcul 1000 fois supérieure à une machine petascale, mais pour un même volume, et donc, une même consommation énergétique, de l'ordre de 20 MW). La technique du "speed-scaling" consiste à modifier la puissance entrante dans certains des processeurs. La diminution de cette puissance a évidemment des conséquence énergétiques: en général on dit que la consommation énergétique du calcul est proportionnelle au carré de cette puissance entrante. Un contrepoint est un impact négatif sur le temps d'exécution des applications : la vitesse d'exécution des tâches se trouve diminuée. De plus, cette diminution a aussi un impact négatif sur leur fiabilité ! Dans ce contexte, nous avons étudié la technique du speed-scaling, couplée à des techniques d'augmentation de fiabilité comme la re-exécution (cette technique consiste à re-exécuter une tâche dont l'exécution a échoué, directement après l'échec sur le même processeur), la réplication (cette technique consiste à exécuter une tâche sur deux processeurs différents en parallèle, à un prix énergétique doublé) ainsi que le checkpoint. Pour ces différents problèmes, nous avons pu fournir des algorithmes dont l'efficacité a été montrée soit au travers de simulations, soit grâce à des preuves d'approximation du résultat par rapport à la solution optimale. Enfin dans un deuxième temps, nous avons considéré, sans speed-scaling, les différents coûts énergétiques dans la technique du checkpoint périodique et coordonné. Nous avons calculé la période optimale pour minimiser cette consommation énergétique et nous l'avons comparée à celle pour minimiser le temps d'exécution.

Introduction

A significant research effort has focused on the characteristics, features, and challenges of High Performance Computing (HPC) systems capable of reaching the Exaflop performance mark [1, 39, 108]. The portrayed Exascale systems will necessitate billion way parallelism, resulting not only in a massive increase in the number of processing units (cores), but also in terms of computing nodes. In order to advance to the next generation of supercomputers, new scientific breakthroughs are needed in computing technology (for example energy-efficient hardware, algorithms, applications, system software).

In February 2014, the Advanced Scientific Computing Advisory Committee (ASCAC) published the top ten Exascale research challenges [1] to achieve the development of an Exascale system. They are sorted into ten categories:

- Energy efficiency: Creating more energy-efficient circuit, power, and cooling technologies.
- Interconnect technology: Increasing the performance and energy efficiency of data movement.
- Memory technology: Integrating advanced memory technologies to improve both capacity and bandwidth.
- Scalable system software: Developing scalable system software that is power- and resilience-aware.
- Programming systems: Inventing new programming environments that express massive parallelism, data locality, and resilience.
- Data management: Creating data management software that can handle the volume, velocity and diversity of data that is anticipated.
- Exascale algorithms: Reformulating science problems and redesigning, or reinventing the algorithms for solving these in Exascale systems.
- Algorithms for discovery, design, and decision: Facilitating mathematical optimization and uncertainty quantification for Exascale discovery, design, and decision making.
- Resilience and correctness: Ensuring correct scientific computation in face of faults, reproducibility, and algorithm verification challenges.
- Scientific productivity: Increasing the productivity of computational scientists with new software engineering tools and environments.

This thesis deals with two issues that appear multiple times in the above list, namely resilience and energy. We address these two issues in two main parts. The first part focuses on the importance of reliability for future Exascale platforms, while the second part discuss how to improve the energy consumption of those platforms.

Considering the relative slopes describing the evolution of the reliability of individual components on one side, and the evolution of the number of components on the other side, the reliability of the entire platform is expected to decrease, due to probabilistic amplification. Even if each independent component is quite reliable, the Mean Time Between Failures (MTBF) is expected to drop drastically. Executions of large parallel applications on these systems will have to tolerate a higher degree of errors and failures than in current systems. The de-facto general-purpose error recovery technique in high performance

computing is checkpoint and rollback recovery. Such protocols employ checkpoints to periodically save the state of a parallel application, so that when an error strikes some process, the application can be restored into one of its former states. The most widely used protocol is coordinated checkpointing, where all processors periodically stop computing and synchronize to write critical application data onto stable storage. Coordinated checkpointing is well understood, at least in its blocking form (when no computing activity takes place during checkpoints), and good approximations of the optimal checkpoint interval exist; they are known as Young's [124] and Daly's [35] formula. We introduce the new reliability problems with further details in Chapter 1, as an introduction to Part I of the thesis. Part I, then, elaborates on the checkpointing methods as a solution to reliability of Exascale systems.

While reliability is a major concern for Exascale, another key challenge is to minimize energy consumption, which we address in Part II of the thesis. The principal reason is that Exascale systems are expected to have similar size as the current Petascale systems, but at an extreme scale capacity, namely 1000 times today's capacity! In order to be able to bring the necessary power to those systems, the thermal power by cm^2 needs to be reduced drastically: as it is now, it is close to the one of a nuclear reactor [71] (Figure 1). In particular, as can be seen on Figure 1, in 2004 an inflexion point was reached: because the thermal density of chips reached that of a nuclear reactor, the industry had to abandon clock frequency growth to avoid melting the chips.

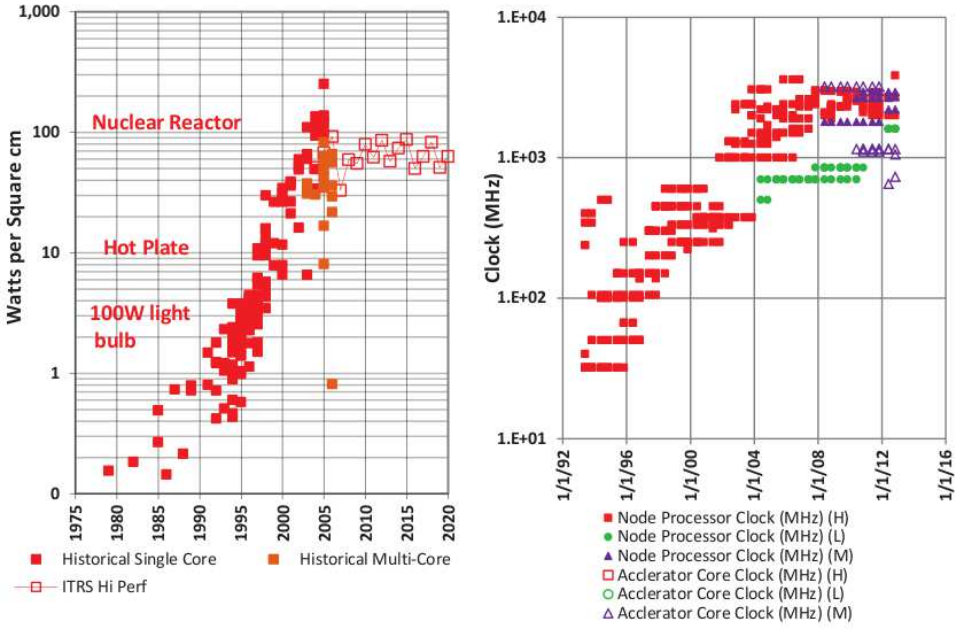


Figure 1: ITRS = International Technology Roadmap for Semiconductors. Source: Kogge and Shalf [71].

On a side note, Horst Simon, the Deputy Director at the Lawrence Berkeley National Laboratory's NERSC (National Energy Research Scientific Computing Center), announced in May 2013 that he bet $2,000 that Exascale will not be reached by 2020 for this reason mainly. This is not the only reason why the focus on the energy performance of future Exascale systems is important. Other reasons include environmental and economical concerns. Reducing power consumption by one megawatt may save around $1M per year even in a relatively inexpensive energy contract [39].

One of the most power-consuming components of today's systems is the processor. Even when idle, it dissipates a significant fraction of the total power. However, for future Exascale systems, the power

dissipated to execute I/O transfers is likely to play an even more important role, because the relative cost of communication is expected to dramatically increase, both in terms of latency and consumed energy [111]. We introduce the energy challenges for Exascale with further details in Chapter 5, as an introduction to Part II of the thesis. Part II, then, delves into methods to improve the energy consumption of Exascale platforms, through various techniques, under a reliability constraint.

We now summarize the different chapters of this thesis. Chapters 1–4 form Part I, and Chapters 5–9 form Part II. We give a concluding chapter for the whole thesis at the end.

Chapter 1: Introduction to coordinated checkpointing

This chapter introduces the part on reliability via periodic checkpointing. Along with new general results, we discuss the related work for Part I.

Chapter 2: Checkpointing algorithms and fault prediction [J4]

This chapter deals with the impact of fault prediction techniques on checkpointing strategies. We extend the classical first-order analysis of Young and Daly in the presence of a fault prediction system, characterized by its recall and its precision. In this framework, we provide optimal algorithms to decide whether and when to take predictions into account, and we derive the optimal value of the checkpointing period. These results allow to analytically assess the key parameters that impact the performance of fault predictors at very large scale.

Chapter 3: Checkpointing strategies with prediction windows [C9]

This chapter deals with the impact of fault prediction techniques on checkpointing strategies. We consider fault-prediction systems that do not provide exact prediction dates, but instead time intervals during which faults are predicted to strike. These intervals dramatically complicate the analysis of the checkpointing strategies. We propose a new approach based upon two periodic modes, a regular mode outside prediction windows, and a proactive mode inside prediction windows, whenever the size of these windows is large enough. We are able to compute the best period for any size of the prediction windows, thereby deriving the scheduling strategy that minimizes platform waste. In addition, the results of the analytical study are nicely corroborated by a comprehensive set of simulations, which demonstrates the validity of the model and the accuracy of the approach.

Chapter 4: On the combination of silent error detection and checkpointing [C5]

In this chapter, we revisit traditional checkpointing and rollback recovery strategies, with a focus on silent data corruption errors. Contrarily to fail-stop failures, such latent errors cannot be detected immediately, and a mechanism to detect them must be provided. We consider two models: (i) errors are detected after some delays following a probability distribution (typically, an Exponential distribution); (ii) errors are detected through some verification mechanism. In both cases, we compute the optimal period in order to minimize the waste, i.e., the fraction of time where nodes do not perform useful computations. In practice, only a fixed number of checkpoints can be kept in memory, and the first model may lead to an irrecoverable failure. In this case, we compute the minimum period required for

an acceptable risk. For the second model, there is no risk of irrecoverable failure, owing to the verification mechanism, but the corresponding overhead is included in the waste. Finally, both models are instantiated using realistic scenarios and application/architecture parameters.

Chapter 5: Introduction to energy-efficient scheduling

In this chapter, we introduce energy-efficient schedules. After an introduction of the Dynamic Voltage and Frequency Scaling (DVFS) technique, we show how to use it through an example of scheduling problem where reliability is not a contraint. We then discuss how energy efficiency impacts reliability and present some related work.

Chapter 6: Energy-aware scheduling under reliability and makespan constraints [C2]

We consider a task graph to be executed on a set of homogeneous processors. We aim at minimizing the energy consumption while enforcing two constraints: a prescribed bound on the execution time (or makespan), and a reliability threshold. Dynamic voltage and frequency scaling (DVFS) is a model frequently used to reduce the energy consumption of a schedule, but it has negative effect on its reliability. In this chapter, to improve the reliability of a schedule while reducing the energy consumption, we allow for the re-execution of some tasks. We assess the complexity of the tri-criteria scheduling problem (makespan, reliability, energy) with two different speed models: either processors can have arbitrary speeds (continuous speeds), or a processor can run at a finite number of different speeds, and it can change its speed during a computation. We propose several novel tri-criteria scheduling heuristics under the continuous speed model, and we evaluate them through a set of simulations. Our two best heuristics turn out to be very efficient and complementary.

Chapter 7: Approximation algorithms for energy, reliability and makespan optimization problems [RR1]

In this chapter, we consider the problem of scheduling an application on a parallel computational platform. The application is a particular task graph, either a linear chain of tasks, or a set of independent tasks. The platform is made of identical processors, whose speed can be dynamically modified. It is also subject to failures: if a processor is slowed down to decrease the energy consumption, it has a higher chance to fail. Therefore, the scheduling problem requires us to re-execute or replicate tasks (i.e., execute twice the same task, either on the same processor, or on two distinct processors), in order to increase the reliability. It is a tri-criteria problem: the goal is to minimize the energy consumption, while enforcing a bound on the total execution time (the makespan), and a constraint on the reliability of each task. Our main contribution is to propose approximation algorithms for these particular classes of task graphs. For linear chains, we design a fully polynomial-time approximation scheme. However, we show that there exists no constant factor approximation algorithm for independent tasks, unless P=NP, and we are able in this case to propose an approximation algorithm with a relaxation on the makespan constraint.

Chapter 8: Energy-aware checkpointing of divisible tasks with soft or hard deadlines [C7]

In this chapter, we aim at minimizing the energy consumption when executing a divisible workload under a bound on the total execution time, while resilience is provided through checkpointing. We discuss several variants of this multi-criteria problem. Given the workload, we need to decide how

many chunks to use, what are the sizes of these chunks, and at which speed each chunk is executed. Furthermore, since a failure may occur during the execution of a chunk, we also need to decide at which speed a chunk should be re-executed in the event of a failure. The goal is to minimize the expectation of the total energy consumption, while enforcing a deadline on the execution time, that should be met either in expectation (soft deadline), or in the worst case (hard deadline). For each problem instance, we propose either an exact solution, or a function that can be optimized numerically. The different models are then compared through an extensive set of experiments.

Chapter 9: Optimal checkpointing period: Time vs. energy [C4]

This short chapter deals with parallel scientific applications using non-blocking and periodic coordinated checkpointing to enforce resilience. We provide a model and detailed formulas for total execution time and consumed energy. We characterize the optimal period for both objectives, and we assess the range of time/energy trade-offs to be made by instantiating the model with a set of realistic scenarios for Exascale systems. We give a particular emphasis to I/O transfers, because the relative cost of communication is expected to dramatically increase, both in terms of latency and consumed energy, for future Exascale platforms.

Part I

Reliability via periodic checkpointing

Chapter 1

Introduction to coordinated periodic checkpointing

Nowadays, the most powerful High Performance Computing (HPC) systems experience about one fault per day [109, 128]. Furthermore, the reliability of an entire platform is expected to decrease, due to probabilistic amplification, as its number of components increases. Indeed, even if each independent component is quite reliable, the Mean Time Between Failures (MTBF) is expected to drop drastically when considering an Exascale system [39]. Failures become a normal part of application executions. Therefore, applications running on large computing systems have to cope with platform faults.

Applications need to use fault-tolerance mechanisms such as checkpoint and rollback, in order to become resilient. Unfortunately, with the advent of Exascale systems, this is not sufficient anymore: with 10,000,000 core processors or more, the time interval between two consecutive failures is anticipated to be smaller than the typical duration of the checkpoint. Recently, considerable research has focused on system administrators trying to predict where and when faults will strike. In particular, most of the research has been devoted to fault predictors [45, 46, 47, 79, 125, 131]. However, no predictor will ever be able to predict every fault. Therefore, fault predictors will have to be used in conjunction with fault-tolerance mechanisms. In Chapters 2 and 3, we discuss efficient techniques to use these fault predictors.

While this fault-tolerant research is important, it assumes instantaneous error detection, and therefore applies to fail-stop failures, such as the crash of a resource. In Chapter 4, we revisit checkpoint protocols in the context of *silent* errors, also called silent data corruption. In HPC, it has been shown recently that such errors are not unusual, and must also be accounted for [90]. The cause may be for instance soft efforts in L1 cache, or double bit flips. The problem is that the detection of a latent error is not immediate, because the error is identified only when the corrupted data is activated. One must then account for the interval required to detect the error in the recovery protocol. Indeed, if the last checkpoint saved an already corrupted state, it may not be possible to recover from the error. Hence the necessity to keep several checkpoints so that one can rollback to the last *correct* state.

Assume that we have jobs executing on a platform subject to faults, and let μ be the Mean Time Between Faults (MTBF) of the platform. In the absence of fault prediction, the standard approach is to take periodic coordinated checkpoints, each of length C, every period of duration T. In steady-state utilization of the platform, the value T_{opt} of T that minimizes the expected waste of resource usage due to checkpointing is approximated as $T_{\text{opt}} = \sqrt{2\mu C} + C$, or $T_{\text{opt}} = \sqrt{2(\mu + D + R)C} + C$ (where D and R are the duration of the down time and of the recovery, respectively). The former expression is the well-known Young's formula [124], while the latter is due to Daly [35]. While there are other types of checkpointing techniques (namely uncoordinated and hierarchical), we focus on coordinated checkpoints. This is because of the fact that they are supported by most of the fault-tolerance libraries

available for HPC [129, 90].

In this introductory chapter, we start with some general results that are useful when considering periodic checkpointing. The first of these results is relative to the MTBF μ of a platform made of N individual components whose individual MTBF is μ_{ind}: $\mu = \frac{\mu_{\text{ind}}}{N}$. The second result is a refined first-order analysis for instantaneous fault detection. When faults follow an exponential distribution, it leads to similar periods than Young [124] and Daly [35], but leads to better performance when faults follow a Weibull distribution. Then we discuss related work in Section 1.2.

1.1 General results

1.1.1 MTBF of a platform

When considering a platform prone to failures, the key parameter is μ, the MTBF of the platform. If the platform is made of N components whose individual MTBF is μ_{ind}, then $\mu = \frac{\mu_{\text{ind}}}{N}$. This result is true regardless of the fault distribution law.

Proposition 1.1. *Consider a platform comprising N components, and assume that the inter-arrival times of the faults on the components are independent and identically distributed random variables that follow an arbitrary probability law whose expectation is finite and $\mu_{ind} > 0$. Then the MTBF μ of the platform (defined as the expectation of the sum of number of failures of the N processors over time), is equal to $\frac{\mu_{ind}}{N}$.*

Proof. Consider first a single component, say component number q. Let X_i, $i \geq 0$ denote the IID random variables for fault inter-arrival times on that component, with $\mathbb{E}(X_i) = \mu_{\text{ind}}$. Consider a fixed time bound F. Let $n_q(F)$ be the number of faults on the component until time F is exceeded. In other words, the $(n_q(F)-1)$-th fault is the last one to happen strictly before time F, and the $n_q(F)$-th fault is the first to happen at time F or after. By definition of $n_q(F)$, we have

$$\sum_{i=1}^{n_q(F)-1} X_i \leq F \leq \sum_{i=1}^{n_q(F)} X_i.$$

Using Wald's equation [105, p. 486], with $n_q(F)$ as a stopping criterion, we derive:

$$(\mathbb{E}(n_q(F)) - 1)\mu_{\text{ind}} \leq F \leq \mathbb{E}(n_q(F))\,\mu_{\text{ind}},$$

and we obtain:

$$\lim_{F \to +\infty} \frac{\mathbb{E}(n_q(F))}{F} = \frac{1}{\mu_{\text{ind}}}. \tag{1.1}$$

Consider now the whole platform, and let Y_i, $i \geq 0$ denote the random variables for fault inter-arrival times on the platform. Let μ, with $\mathbb{E}(Y_i) = \mu$. Consider a fixed time bound F as before. Let $n(F)$ be the number of faults on the whole platform until time F is exceeded. Let $m_q(F)$ be the number of these faults that strike component number q. Of course we have $n(F) = \sum_{q=1}^{N} m_q(F)$. By definition, except for the component hit by the last failure, $m_q(F)+1$ is the number of failures on component q until time F is exceeded, hence $n_q(F) = m_q(F) + 1$ (and this number is $m_q(F) = n_q(F)$ on the component hit by the last failure). From Equation (1.1) again, we have for each component q:

$$\lim_{F \to +\infty} \frac{\mathbb{E}(m_q(F))}{F} = \frac{1}{\mu_{\text{ind}}}.$$

Since $n(F) = \sum_{q=1}^{N} m_q(F)$, we also have:

$$\lim_{F \to +\infty} \frac{\mathbb{E}\left(n(F)\right)}{F} = \frac{N}{\mu_{\text{ind}}}. \tag{1.2}$$

■

The random variables Y_i are not IID, but another possible asymptotic definition of the MTBF μ of the platform could be $\frac{1}{\mu} = \lim_{F \to +\infty} \frac{\sum_i^{n(F)} Y_i}{F}$. Kella and Stadje (Theorem 4, [69]) proved that this limit indeed exists and that μ is also equal to $\frac{\mu_{\text{ind}}}{N}$, if in addition the distribution function of the X_i is continuous.

These two results do not give the expectation of each random variable Y_i. The simplest case is that of the occurrence of the first error Y_1. Indeed, if we know the probability distribution F_X of each component, then Y_1 is the minimum of the random variables corresponding to the first error on each processor, hence its distribution law is given by $F_{Y_1}(t) = 1 - (1 - F_X(t))^N$. However, the expectation of the arrival times of the next failures are much more complicated to derive, because they depend upon the history of the platform.

1.1.2 Revisiting Daly's first-order approximation

Young proposed a "first order approximation to the optimum checkpoint interval" [124]. Young's formula was later refined by Daly [35] to take into account the recovery time. We revisit their analysis using the notion of waste. We remind that in the following, C is the time to execute a checkpoint, D is the duration of a down time, and R is the duration of the recovery of a checkpoint (following a down time).

Let $\text{TIME}_{\text{base}}$ be the base time of the application without any overhead (neither checkpoints nor faults). First, assume a *fault-free* execution of the application with periodic checkpointing. In such an environment, during each period of length T we take a checkpoint, which lasts for C time units, and only $T - C$ units of work are executed. Let TIME_{FF} be the execution time of the application in this setting. Following most work in the literature, we also take a checkpoint at the end of the execution. The fault-free execution time TIME_{FF} is equal to the time needed to execute the whole application, $\text{TIME}_{\text{base}}$, plus the time taken by the checkpoints:

$$\text{TIME}_{\text{FF}} = \text{TIME}_{\text{base}} + N_{\text{ckpt}} C, \tag{1.3}$$

where N_{ckpt} is the number of checkpoints taken. We have

$$N_{\text{ckpt}} = \left\lceil \frac{\text{TIME}_{\text{base}}}{T - C} \right\rceil \approx \frac{\text{TIME}_{\text{base}}}{T - C}.$$

When discarding the ceiling function, we assume that the execution time is very large with respect to the period or, symmetrically, that there are many periods during the execution. Plugging back the (approximated) value $N_{\text{ckpt}} = \frac{\text{TIME}_{\text{base}}}{T-C}$, we derive that

$$\text{TIME}_{\text{FF}} = \frac{\text{TIME}_{\text{base}}}{T - C} T. \tag{1.4}$$

The waste due to checkpointing in a fault-free execution, WASTE_{FF}, is defined as the fraction of the execution time that does not contribute to the progress of the application:

$$\text{WASTE}_{\text{FF}} = \frac{\text{TIME}_{\text{FF}} - \text{TIME}_{\text{base}}}{\text{TIME}_{\text{FF}}} \qquad \Leftrightarrow \qquad \left(1 - \text{WASTE}_{\text{FF}}\right) \text{TIME}_{\text{FF}} = \text{TIME}_{\text{base}}. \tag{1.5}$$

Combining Equations (1.4) and (1.5), we get:

$$\text{WASTE}_{\text{FF}} = \frac{C}{T}. \tag{1.6}$$

Now, let $\text{TIME}_{\text{final}}$ denote the expected execution time of the application in the presence of faults. This execution time can be divided into two parts: (i) the execution of "chunks" of work of size $T - C$ followed by their checkpoint; and (ii) the time lost due to the faults. This decomposition is illustrated in Figure 1.1. The first part of the execution time is equal to TIME_{FF}. Let N_{faults} be the number of faults occurring during the execution, and let T_{lost} be the average time lost per fault. Then,

$$\text{TIME}_{\text{final}} = \text{TIME}_{\text{FF}} + N_{\text{faults}} T_{\text{lost}}. \tag{1.7}$$

On average, during a time $\text{TIME}_{\text{final}}$, $N_{\text{faults}} = \frac{\text{TIME}_{\text{final}}}{\mu}$ faults happen. We need to estimate T_{lost}. A natural estimation of the computation loss is $\frac{T}{2}$ and has been proven by Daly [35] for exponential laws. We use it in the general case as an approximation. We conclude that $T_{\text{lost}} = \frac{T}{2} + D + R$, because after each fault there is a downtime and a recovery. This leads to:

$$\text{TIME}_{\text{final}} = \text{TIME}_{\text{FF}} + \frac{\text{TIME}_{\text{final}}}{\mu}\left(D + R + \frac{T}{2}\right).$$

Let $\text{WASTE}_{\text{fault}}$ be the fraction of the total execution time that is lost because of faults:

$$\text{WASTE}_{\text{fault}} = \frac{\text{TIME}_{\text{final}} - \text{TIME}_{\text{FF}}}{\text{TIME}_{\text{final}}} \qquad \Leftrightarrow \qquad (1 - \text{WASTE}_{\text{fault}})\,\text{TIME}_{\text{final}} = \text{TIME}_{\text{FF}} \tag{1.8}$$

We derive:

$$\text{WASTE}_{\text{fault}} = \frac{1}{\mu}\left(D + R + \frac{T}{2}\right). \tag{1.9}$$

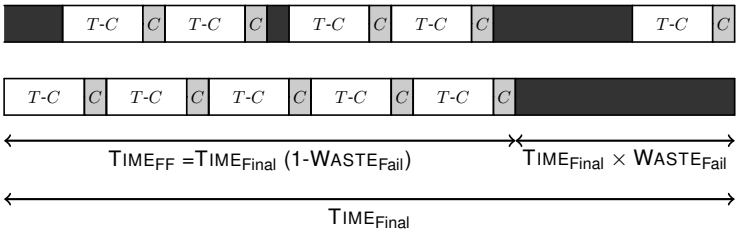


Figure 1.1: An execution (top), and its re-ordering (bottom), to illustrate both sources of waste. Blackened intervals correspond to work destroyed by faults, downtimes, and recoveries.

Daly [35] uses the expression

$$\text{TIME}_{\text{final}} = \big(1 + \text{WASTE}_{\text{fault}}\big)\text{TIME}_{\text{FF}} \tag{1.10}$$

instead of Equation (1.8), which leads him to his well-known first-order formula

$$T = \sqrt{2(\mu + (D + R))C} + C. \tag{1.11}$$

Figure 1.1 explains why Equation (1.10) is not correct and should be replaced by Equation (1.8). Indeed, the expected number of faults depends on the final time, not on the time for a fault-free execution. We point out that Young [124] also used Equation (1.10), but with $D = R = 0$. Equation (1.8) can be

rewritten $\text{TIME}_{\text{final}} = \text{TIME}_{\text{FF}}/\left(1 - \text{WASTE}_{\text{fault}}\right)$. Therefore, using Equation (1.10) instead of Equation (1.8), in fact, is equivalent to writing $\frac{1}{1-\text{WASTE}_{\text{fault}}} \approx 1 + \text{WASTE}_{\text{fault}}$ which is indeed a first-order approximation if $\text{WASTE}_{\text{fault}} \ll 1$.

Now, let WASTE denote the total waste:

$$\text{WASTE} = \frac{\text{TIME}_{\text{final}} - \text{TIME}_{\text{base}}}{\text{TIME}_{\text{final}}}. \tag{1.12}$$

Then,

$$\text{WASTE} = 1 - \frac{\text{TIME}_{\text{base}}}{\text{TIME}_{\text{final}}} = 1 - \frac{\text{TIME}_{\text{base}}}{\text{TIME}_{\text{FF}}} \frac{\text{TIME}_{\text{FF}}}{\text{TIME}_{\text{final}}} = 1 - (1 - \text{WASTE}_{\text{FF}})(1 - \text{WASTE}_{\text{fault}}).$$

Altogether, we derive the final result:

$$\begin{aligned} \text{WASTE} &= \text{WASTE}_{\text{FF}} + \text{WASTE}_{\text{fault}} - \text{WASTE}_{\text{FF}}\text{WASTE}_{\text{fault}} & (1.13)\\ &= \frac{C}{T} + \left(1 - \frac{C}{T}\right)\frac{1}{\mu}\left(D + R + \frac{T}{2}\right). & (1.14) \end{aligned}$$

We obtain $\text{WASTE} = \frac{u}{T} + v + wT$, where $u = C\left(1 - \frac{D+R}{\mu}\right)$, $v = \frac{D+R-C/2}{\mu}$, and $w = \frac{1}{2\mu}$. Thus simple algebra show that WASTE is minimized for $T = \sqrt{\frac{u}{w}}$. The Refined First-Order (RFO) formula for the optimal period is thus:

$$T_{\text{RFO}} = \sqrt{2(\mu - (D + R))C}. \tag{1.15}$$

It is interesting to point out why Equation (1.15) is a first-order approximation, even for large jobs. Indeed, there are several restrictions for the approach to be valid:

- We have stated that the expected number of faults during execution is $N_{\text{faults}} = \frac{\text{TIME}_{\text{final}}}{\mu}$, and that the expected time lost due to a fault is $T_{\text{lost}} = \frac{T}{2} + D + R$. Both statements are true individually, but the expectation of a product is the product of the expectations only if the random variables are independent, which is not the case here because $\text{TIME}_{\text{final}}$ depends upon the failure inter-arrival times.
- We have used that the computation time lost when a failure happens is $\frac{T}{2}$ which has been proven true for exponential and uniform distributions only.
- In Equation (1.6), we have to enforce $C \leq T$ to have $\text{WASTE}_{\text{FF}} \leq 1$.
- In Equation (1.9), we have to enforce $D + R \leq \mu$ and bound T in order to have $\text{WASTE}_{\text{fault}} \leq 1$. Intuitively, we need μ to be large enough for Equation (1.9) to make sense. However, regardless of the value of the individual MTBF μ_{ind}, there is always a threshold in the number of components N above which the platform MTBF, $\mu = \frac{\mu_{\text{ind}}}{N}$, becomes too small for Equation (1.9) to be valid.
- Equation (1.9) is accurate only when two or more faults do not take place within the same period. Although unlikely when μ is large in front of T, the possible occurrence of many faults during the same period cannot be eliminated.

To ensure that the latter condition (at most a single fault per period) is met with a high probability, we cap the length of the period: we enforce the condition $T \leq \alpha\mu$, where α is some tuning parameter chosen as follows. The number of faults during a period of length T can be modeled as a Poisson process of parameter $\beta = \frac{T}{\mu}$. The probability of having $k \geq 0$ faults is $P(X = k) = \frac{\beta^k}{k!}e^{-\beta}$, where X is the random variable showing the number of faults. Hence the probability of having two or more faults is $\pi = P(X \geq 2) = 1 - (P(X = 0) + P(X = 1)) = 1 - (1 + \beta)e^{-\beta}$. If we assume $\alpha = 0.27$ then $\pi \leq 0.03$, hence a valid approximation when bounding the period range accordingly. Indeed, with such a conservative value for α, we have overlapping faults for only 3% of the checkpointing segments in

average, so that the model is quite reliable. For consistency, we also enforce the same type of bound on the checkpoint time, and on the downtime and recovery: $C \leq \alpha\mu$ and $D+R \leq \alpha\mu$. However, enforcing these constraints may lead to use a sub-optimal period: it may well be the case that the optimal period $\sqrt{2(\mu-(D+R))C}$ of Equation (1.15) does not belong to the admissible interval $[C, \alpha\mu]$. In that case, the waste is minimized for one of the bounds of the admissible interval. This is because, as seen from Equation (1.14), the waste is a convex function of the period.

We conclude this discussion on a positive note. While capping the period, and enforcing a lower bound on the MTBF, is mandatory for mathematical rigor, simulations (see Chapter 2 for both exponential and Weibull distributions) show that actual job executions can always use the value from Equation (1.15), accounting for multiple faults whenever they occur by re-executing the work until success. The first-order model turns out to be surprisingly robust!

To the best of our knowledge, despite all the limitations above, there is no better approach to estimate the waste due to checkpointing when dealing with arbitrary fault distributions. However, assuming that the faults obey an exponential distribution, it is possible to use the memory less property of this distribution to provide more accurate results. A second-order approximation when faults obey an exponential distribution is given by Daly [35, Equation (20)] as $\text{TIME}_{\text{final}} = \mu e^{R/\mu}(e^{\frac{T}{\mu}} - 1)\frac{\text{TIME}_{\text{base}}}{T-C}$. In fact, in that case, the exact value of $\text{TIME}_{\text{final}}$ is $\text{TIME}_{\text{final}} = (\mu+D)e^{R/\mu}(e^{\frac{T}{\mu}} - 1)\frac{\text{TIME}_{\text{base}}}{T-C}$ [17, 104], and the optimal period is then $\frac{1+\mathbb{L}(-e^{-\frac{C}{\mu}-1})}{\mu}$ where $\mathbb{L}$, the Lambert function, is defined as $\mathbb{L}(z)e^{\mathbb{L}(z)} = z$.

To assess the accuracy of the different first order approximations, we compare the periods defined by Young's formula [124], Daly's formula [35], and Equation (1.15), to the optimal period, in the case of an exponential distribution. Results are reported in Table 1.1. To establish these results, we use parameters realistic for current and future platforms: $C = R = 600$ s, $D = 60$ s, and $\mu_{\text{ind}} = 125$ years. One can observe in Table 1.1 that the relative error for Daly's period is slightly larger than the one for Young's period. In turn, the absolute value of the relative error for Young's period is slightly larger than the one for RFO. More importantly, when Young's and Daly's formulas overestimate the period, RFO underestimates it. Table 1.1 does not allow us to assess whether these differences are actually significant. However, we will report in Chapter 2 some simulations that show that Equation (1.15) leads to smaller execution times for Weibull distributions than both classical formulas (Tables 2.3 and 2.4).

N	μ	YOUNG		DALY		RFO		Optimal
2^{10}	3849609	68567	(0.5 %)	68573	(0.5 %)	67961	(-0.4 %)	68240
2^{11}	1924805	48660	(0.7 %)	48668	(0.7 %)	48052	(-0.6 %)	48320
2^{12}	962402	34584	(1.2 %)	34595	(1.2 %)	33972	(-0.6 %)	34189
2^{13}	481201	24630	(1.6 %)	24646	(1.7 %)	24014	(-0.9 %)	24231
2^{14}	240601	17592	(2.3 %)	17615	(2.5 %)	16968	(-1.3 %)	17194
2^{15}	120300	12615	(3.2 %)	12648	(3.5 %)	11982	(-1.9 %)	12218
2^{16}	60150	9096	(4.5 %)	9142	(5.1 %)	8449	(-2.9 %)	8701
2^{17}	30075	6608	(6.3 %)	6673	(7.4 %)	5941	(-4.4 %)	6214
2^{18}	15038	4848	(8.8 %)	4940	(10.8 %)	4154	(-6.8 %)	4458
2^{19}	7519	3604	(12.0 %)	3733	(16.0 %)	2869	(-10.8 %)	3218

Table 1.1: Comparing periods produced by the different approximations with optimal value. Beside each period, we report its relative deviation from the optimal. Each value is expressed in seconds.

1.2 Related work

Traditional (coordinated) checkpointing has been studied for many years. The major appeal of the coordinated approach is its simplicity, because a parallel job using n processors of individual MTBF μ_{ind} can be viewed as a single processor job with MTBF $\mu = \frac{\mu_{ind}}{n}$ (as was shown in Section 1.1.1). Given the value of μ, an approximation of the optimal checkpointing period can be computed as a function of the key parameters (down time D, checkpoint time C, and recovery time R). As already mentioned, the first estimate had been given by Young [124] and later refined by Daly [35]. Both use a first-order approximation for exponential failure distributions. More accurate formulas for Weibull failure distributions are provided by some authors [18, 80, 97]. The optimal checkpointing period is known only for exponential failure distributions [17]. Dynamic programming heuristics for arbitrary distributions are proposed [116, 19, 17]. Gelenbe and Derochette [51] give a first-order approximation of the optimal period to minimize the average response time. They compare it to the period obtained by Young [124] in a model where they do not consider one single long application and a system fully-loaded, but instead multiple small independent applications that arrive in the system following a Poisson process. In this context, Gelenbe [50] shows that the optimal period for the waste depends on the load of the system. Finally, Gelenbe and Hernández [52] compute the optimal checkpoint interval for the waste with age dependent failures: they assume that the failure rate follows a Weibull distribution and that each checkpoint in a renewal point.

The literature proposes different studies [67, 94, 98, 117, 130] on the modeling of coordinated checkpointing protocols. For instance, Jin et al. [67] and Plank and Thomason [98] focus on the usage of available resources: some may be kept as backup in order to replace the down ones, and others may be even shuted down in order to decrease the failure risk or to prevent storage consumption by saving fewer checkpoint snapshots.

The major drawback of coordinated checkpointing protocols is their lack of scalability at extreme-scale. These protocols will lead to I/O congestion when too many processes are checkpointing at the same time. Even worse, transferring the whole memory footprint of an HPC application onto stable storage may well take so much time that a failure is likely to take place during the transfer! A few papers [130, 25] propose a scalability study to assess the impact of a small MTBF (i.e., of a large number of processors). The mere conclusion is that checkpoint time should be either dramatically reduced for platform waste to become acceptable, which motivated the instantiation of optimistic scenarios in Section 4.4, or that checkpointing should be coupled with a system that can predict when a fault may occurs, which motivated Chapters 2 and 3.

Fault prediction. Considerable research has been devoted to fault prediction, using very different models (system log analysis [125], event-driven approach [46, 125, 131], support vector machines [79, 45], nearest neighbors [79], etc). In this section, we give a brief overview of existing predictors, focusing on their characteristics rather than on the methods of prediction. For the sake of clarity, we sum up the characteristics of the different fault predictors from the literature in Table 1.2.

A predictor is characterized by two critical parameters, its recall r, which is the fraction of faults that are indeed predicted, and its precision p, which is the fraction of predictions that are correct (i.e., correspond to actual faults).

Zheng et al. [131] introduce the *lead time*, that is the duration between the time the prediction is made and the time the predicted fault is supposed to happen. This time should be sufficiently large to enable proactive actions. The distribution of lead times is irrelevant. Indeed, only predictions whose lead time is greater than C_p (the time to take a proactive checkpoint) are meaningful. Predictions whose lead time is smaller than C_p, whenever they materialize as actual faults, should be classified as unpredicted

Paper	Lead Time	Precision (p)	Recall (r)	Prediction Window
[131]	300 s	40 %	70%	-
[131]	600 s	35 %	60%	-
[125]	2h	64.8 %	65.2%	yes (size unknown)
[125]	0 min	82.3 %	85.4 %	yes (size unknown)
[46]	32 s	93 %	43 %	-
[47]	10s	92 %	40 %	-
[47]	60s	92 %	20 %	-
[47]	600s	92 %	3 %	-
[45]	NA	70 %	75 %	-
[79]	NA	20 %	30 %	1h
[79]	NA	30 %	75 %	4h
[79]	NA	40 %	90 %	6h
[79]	NA	50 %	30 %	6h
[79]	NA	60 %	85%	12h

Table 1.2: Comparative study of different parameters returned by some predictors.

faults; the predictor recall should be decreased accordingly.

One predictor [131] is also able to locate where the predicted fault is supposed to strike. This additional characteristics has a negative impact on the precision (because a fault happening at the predicted time but not on the predicted location is classified as a non predicted fault; see the low value of p for predictor [131] in Table 1.2). Zheng et al. [131] state that fault localization has a positive impact on proactive checkpointing time in their context: instead of a full checkpoint costing $1,500$ seconds, they can take a partial checkpoint costing only 12 seconds. This led us to introduce a different cost C_p for proactive checkpoints, which can be smaller than the cost C of regular checkpoints. Gainaru et al. [47] also state that fault-localization could help decrease the checkpointing time. Their predictor also gives information on fault localization. They studied the impact of different lead times on the recall of their predictor.

Yu et al. [125] also consider a lead time, and introduce a *prediction window* indicating when the predicted fault should happen. Liang et al. [79] study the impact of different prediction techniques with different prediction window sizes. They also consider a lead time, but do not state its value. These two latter studies motivate our work in Chapter 3, even though Yu et al. [125] do not provide the size of their prediction window.

Most studies on fault prediction state that a proactive action must be taken right before the predicted fault, be it a checkpoint or a migration. However, we show in Chapter 2 that it is beneficial to ignore some predictions, namely when the predicted fault is announced to strike less than $\frac{C_p}{p}$ seconds after the last periodic checkpoint.

Unfortunately, much of the work done on prediction does not provide information that could be really useful for the design of efficient algorithms. Missing information includes the lead time and the size of the prediction window. Other information that could be useful would be: (i) the distribution of the faults in the prediction window; and (ii) the precision and recall as functions of the size of the prediction window (what happens with a larger prediction window). In the simpler case where predictions are exact-date predictions, Gainaru et al. [47] and Bouguerra et al. [20] have shown that the optimal checkpointing period becomes $T_{\text{opt}} = \sqrt{\frac{2\mu C}{1-r}}$, but their analysis is valid only if μ is very large in front of the other parameters, and their computation of the waste is not fully accurate [RR9]. In Chapter 2, we

have refined the results of Gainaru et al. [47], focusing on a more accurate analysis of fault prediction with exact dates, and providing a detailed study on the impact of recall and precision on the waste. As shown in Section 3.3, the analysis of the waste is dramatically more complicated when using prediction windows than when using exact-date predictions.

Li et al. [78] considered the mathematical problem of when and how to migrate. In order to be able to use migration, they assumed that at any time 2% of the resources are available as spares. This allows them to conceive a Knapsack-based heuristic. Thanks to their algorithm, they were able to save 30% of the execution time compared to a heuristic that does not take the prediction into account, with a precision and recall of 70%, and with a maximum load of 70%. In our study, we do not consider that we have a batch of spare resources. We assume that after a downtime the resources that failed are once again available.

Error detection. All the above approaches maintain a single checkpoint. If the checkpoint file includes errors, the application faces an irrecoverable failure and must restart from scratch. This is because error detection latency is ignored in traditional rollback and recovery schemes. These schemes assume instantaneous error detection (therefore mainly targeting fail-stop failures) and are unable to accommodate silent errors.

Considerable efforts have been directed at error-checking to reveal latent errors. Error detection is usually very costly. Hardware mechanisms, such as Error Correcting Code memory (ECC Memory), can detect and even correct a fraction of errors, but in practice they are complemented with software techniques. The simplest technique is triple modular redundancy and voting [83]. For high-performance scientific applications, process replication (each process is equipped with a replica, and messages are quadruplicated) is proposed in the RedMPI library [44]. Application-specific information can be very useful to enable ad-hoc solutions, that dramatically decrease the cost of detection. Many techniques have been advocated. They include memory scrubbing [64], but also ABFT techniques [63, 15, 112], such as coding for the sparse-matrix vector multiplication kernel [112], and coupling a higher-order with a lower-order scheme for PDEs [11]. These methods can only detect an error but do not correct it. Self-stabilizing corrections after error detection in the conjugate gradient method are investigated by Sao and Vuduc [107]. See also Heroux and Hoemmen [60] for the design of a fault-tolerant GMRES capable of converging despite silent errors, and Bronevetsky and de Supinski [23] for a comparative study of detection costs for iterative methods. Lu et al. [82] give a comprehensive list of techniques and references. Our work is agnostic of the underlying error-detection technique and takes the cost of verification as an input parameter to the model (see Section 4.3).

Chapter 2

Checkpointing algorithms and fault prediction

2.1 Introduction

In this chapter, we assess the impact of fault prediction techniques on checkpointing strategies. When some fault prediction mechanism is available, can we compute a better checkpointing period to decrease the expected waste? To what extent? Critical parameters that characterize a fault prediction system are its recall r, which is the fraction of faults that are indeed predicted, and its precision p, which is the fraction of predictions that are correct (i.e., correspond to actual faults). The major objective of this chapter is to refine the expression of the expected waste as a function of these parameters, and to design efficient checkpointing policies that take predictions into account.

Main contributions. In this chapter, we extend the analysis presented in Chapter 1 to fault predictions, and the design of new checkpointing policies that makes optimal decisions on whether and when to take these predictions into account. For policies where the decision to trust the predictor is taken with the same probability throughout the checkpointing period, we show that we should always trust the predictor, or never, depending upon platform and predictor parameters. For policies where the decision to trust the predictor is taken with variable probability during the checkpointing period, we show that we should change strategy only once in the period. We should move from never trusting the predictor when the prediction arrives in the beginning of the period, to always trusting the predictor when the prediction arrives later on in the period. We determine the optimal break-even point. For all policies, we compute the optimal value of the checkpointing period thereby designing optimal algorithms to minimize the waste when coupling checkpointing with predictions.

Finally, we present an extensive set of simulations that corroborates all mathematical derivations. These simulations are based on synthetic fault traces (for exponential fault distributions, and for more realistic Weibull fault distributions) and on log-based fault traces. In addition, they include exact prediction dates and uncertainty intervals for these dates. The section on uncertain intervals is thoroughly studied in Chapter 3.

The rest of the chapter is organized as follows. We first detail the framework in Section 2.2. We provide optimal algorithms to account for predictions in Section 2.3. We start with simpler policies where the decision to trust the predictor is taken with the same probability throughout the checkpointing period (Section 2.3.1) before dealing with the most general approach where the decision to trust the predictor is taken with variable probability during the checkpointing period (Section 2.3.2). Section 2.4 is devoted to simulations: we first describe the simulation framework (Section 2.4.1) and then discuss synthetic and

log-based failure traces in Sections 2.4.2 and 2.4.3 respectively. Finally, we provide concluding remarks in Section 2.5.

2.2 Framework

We summarize the notations that we introduce in this section in Table 2.1.

2.2.1 Checkpointing strategy

We consider a *platform* subject to faults. Our work is agnostic of the granularity of the platform, which may consist either of a single processor, or of several processors that work concurrently and use coordinated checkpointing. *Checkpoints* are taken at regular intervals, or periods, of length T. We denote by C the duration of a checkpoint (all checkpoints have same duration). By construction, we must enforce that $C \leq T$. When a fault strikes the platform, the application is lacking some resource for a certain period of time of length D, the *downtime*. The downtime accounts for software rejuvenation (i.e., rebooting [72, 27]) or for the replacement of the failed hardware component by a spare one. Then, the application recovers from the last checkpoint. R denotes the duration of this *recovery* time.

2.2.2 Fault predictor

A fault predictor is a mechanism that is able to predict that some faults will take place, either at a certain point in time, or within some time-interval window. In this chapter, we assume that the predictor is able to provide exact prediction dates, and to generate such predictions early enough so that a *proactive* checkpoint can indeed be taken before the event.

The accuracy of the fault predictor is characterized by two quantities, the *recall* and the *precision*. The recall r is the fraction of faults that are predicted while the precision p is the fraction of fault predictions that are correct. Traditionally, one defines three types of *events*: (i) *True positive* events are faults that the predictor has been able to predict (let $True_P$ be their number); (ii) *False positive* events are fault predictions that did not materialize as actual faults (let $False_P$ be their number); and (iii) *False negative* events are faults that were not predicted (let $False_N$ be their number). With these definitions, we have $r = \frac{True_P}{True_P + False_N}$ and $p = \frac{True_P}{True_P + False_P}$.

Proactive checkpoints may have a different length C_p than regular checkpoints of length C. In fact there are many scenarios. On the one hand, we may well have $C_p > C$ in scenarios where regular checkpoints are taken at time-steps where the application memory footprint is minimal [61]; on the contrary, proactive checkpoints are taken according to predictions that can take place at arbitrary instants. On the other hand, we may have $C_p < C$ in other scenarios [131], e.g., when the prediction is localized to a particular resource subset, hence allowing for a smaller volume of checkpointed data.

To keep full generality, we deal with two checkpoint sizes in this chapter: C for periodic checkpoints, and C_p for proactive checkpoints (those taken upon predictions).

In the literature, the *lead time* is the interval between the date at which the prediction is made available, and the actual prediction date. While the lead time is an important parameter, the shape of its distribution law is irrelevant to the problem: either a fault is predicted at least C_p seconds in advance, and then one can checkpoint just in time before the fault, or the prediction is useless! In other words, predictions that come too late should be classified as unpredicted faults whenever they materialize as actual faults, leading to a smaller value of the predictor recall.

2.2.3 Fault rates

In addition to μ, the platform MTBF (see Section 1.1.1), let μ_{P} be the mean time between predicted events (both true positive and false positive), and let μ_{NP} be the mean time between unpredicted faults (false negative). Finally, we define the mean time between events as μ_{e} (including all three event types). The relationships between μ, μ_{P}, μ_{NP}, and μ_{e} are the following:

- Rate of unpredicted faults: $\frac{1}{\mu_{\text{NP}}} = \frac{1-r}{\mu}$, since $1-r$ is the fraction of faults that are unpredicted;
- Rate of predicted faults: $\frac{r}{\mu} = \frac{p}{\mu_{\text{P}}}$, since r is the fraction of faults that are predicted, and p is the fraction of fault predictions that are correct;
- Rate of events: $\frac{1}{\mu_{\text{e}}} = \frac{1}{\mu_{\text{P}}} + \frac{1}{\mu_{\text{NP}}}$, since events are either predictions (true or false), or unpredicted faults.

2.2.4 Objective: Waste minimization

The natural objective is to minimize the expectation of the total execution time, *makespan*, of the application. Instead, in order to ease mathematical derivations, we aim at minimizing the *waste*. The waste is the expected percentage of time lost, or "wasted", during the execution. In other words, the *waste* is the fraction of time during which the platform is not doing useful work. This definition was introduced by Wingstrom [120]. Obviously, the lower the waste, the lower the expected makespan, and reciprocally. Hence the two objectives are strongly related and minimizing one of them also minimizes the other.

p	Predictor precision: proportion of true positives among the number of predicted faults
r	Predictor recall: proportion of predicted faults among total number of faults
q	Probability to trust the predictor
MTBF	Mean Time Between Faults
N	Number of processors in the platform
μ	Platform MTBF
μ_{ind}	Individual MTBF
μ_{P}	Rate of predicted faults
μ_{NP}	Rate of unpredicted faults
μ_{e}	Rate of events (predictions or unpredicted faults)
D	Downtime
R	Recovery time
C	Duration of a regular checkpoint
C_p	Duration of a proactive checkpoint
T	Duration of a period

Table 2.1: Table of main notations.

2.3 Taking predictions into account

In this section, we present an analytical model to assess the impact of predictions on periodic checkpointing strategies. As already mentioned, we consider the case where the predictor is able to provide exact prediction dates, and to generate such predictions at least C_p seconds in advance, so that a proactive checkpoint of length C_p can indeed be taken before the event.

For the sake of clarity, we start with a simple algorithm (Section 2.3.1) which we refine in Section 2.3.2. We then compute the value of the period that minimizes the waste in Section 2.3.3.

2.3.1 Simple policy

In this section, we consider the following algorithm:

- While no fault prediction is available, checkpoints are taken periodically with period T;
- When a fault is predicted, there are two cases: either there is the possibility to take a proactive checkpoint, or there is not enough time to do so, because we are already checkpointing (see Figures 2.1(b) and 2.1(c)). In the latter case, there is no other choice than ignoring the prediction. In the former case, we still have the possibility to ignore the prediction, but we may also decide to trust it: we do this randomly. With probability q, we trust the predictor and take the prediction into account (see Figures 2.1(f) and 2.1(g)), and with probability $1-q$, we ignore the prediction (see Figures 2.1(d) and 2.1(e));
- If we take the prediction into account, we take a proactive checkpoint (of length C_p) as late as possible, so that it completes right at the time when the fault is predicted to happen. After this checkpoint, we complete the execution of the period (see Figures 2.1(f) and 2.1(g));
- If we ignore the prediction, either by necessity (not enough time to take an extra checkpoint, see Figures 2.1(b) and 2.1(c)), or by choice (with probability $1-q$, Figures 2.1(d) and 2.1(e)), we finish the current period and start a new one.

The rationale for not always trusting the predictor is to avoid taking useless checkpoints too frequently. Intuitively, the precision p of the predictor must be above a given threshold for its usage to be worthwhile. In other words, if we decide to checkpoint just before a predicted event, either we will save time by avoiding a costly re-execution if the event does correspond to an actual fault, or we will lose time by unduly performing an extra checkpoint. We need a larger proportion of the former cases, i.e., a good precision, for the predictor to be really useful. The following analysis will determine the optimal value of q as a function of the parameters C, C_p, μ, r, and p.

We could refine the approach by taking into account the amount of work already done in the current period when deciding whether to trust the predictor or not. Intuitively, the more work already done, the more important to save it, hence the more worthwhile to trust the predictor. We design such a refined strategy in Section 2.3.2. Right now, we analyze a simpler algorithm where we decide to trust or not to trust the predictor, independently of the amount of work done so far within the period.

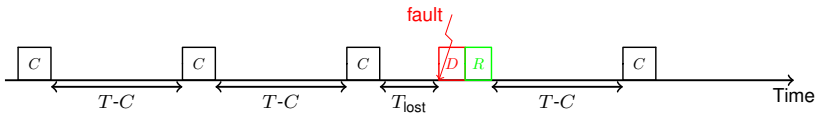
We analyze the algorithm in order to compute a formula for the expected waste, just as in Equation (1.13) (which we remind here):

$$\text{WASTE} = \text{WASTE}_{\text{FF}} + \text{WASTE}_{\text{fault}} - \text{WASTE}_{\text{FF}}\text{WASTE}_{\text{fault}} \tag{2.1}$$

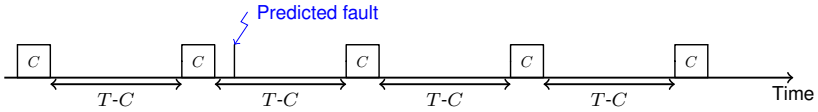
While the value of WASTE_{FF} is unchanged ($\text{WASTE}_{\text{FF}} = \frac{C}{T}$), the value of $\text{WASTE}_{\text{fault}}$ is modified because of predictions. As illustrated in Figure 2.1, there are many different scenarios that contribute to $\text{WASTE}_{\text{fault}}$ that can be sorted into three categories:

(1) **Unpredicted faults:** This overhead occurs each time an unpredicted fault strikes, that is, on average, once every μ_{NP} seconds. Just as in Equation (1.9), the corresponding waste is $\frac{1}{\mu_{\text{NP}}}\left[\frac{T}{2} + D + R\right]$.

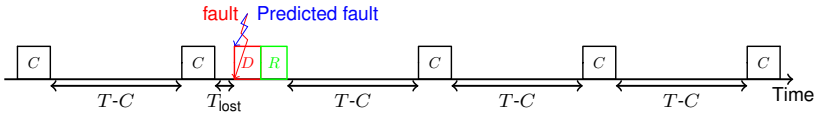
(2) **Predictions not taken into account:** The second source of waste is for predictions that are ignored. This overhead occurs in two different scenarios. First, if we do not have time to take a proactive



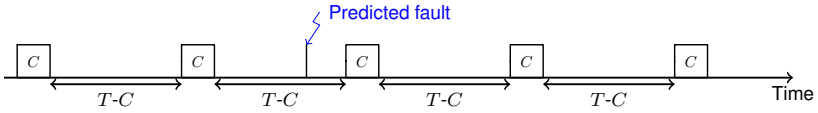
(a) Unpredicted fault



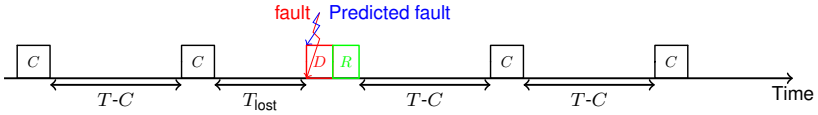
(b) Prediction cannot be taken into account - no actual fault



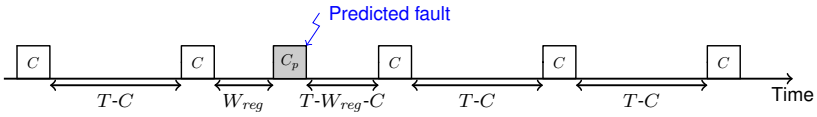
(c) Prediction cannot be taken into account - with actual fault



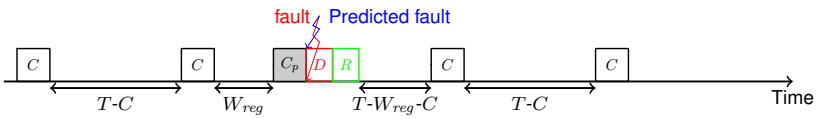
(d) Prediction not taken into account by choice - no actual fault



(e) Prediction not taken into account by choice - with actual fault



(f) Prediction taken into account - no actual fault



(g) Prediction taken into account - with actual fault

Figure 2.1: Actions taken for the different event types.

checkpoint, we have an overhead if and only the prediction is an actual fault. This case happens with probability p. We then lose a time $t + D + R$ if the predicted fault happens a time t after the completion of the last periodic checkpoint. The expected time lost is thus

$$T^1_{\text{lost}} = \frac{1}{T} \int_0^{C_p} (p(t + D + R) + (1 - p)0)\, dt$$

Then, if we do have time to take a proactive checkpoint but still decide to ignore the prediction, we also have an overhead if and only the prediction is an actual fault, but the expected time lost is now weighted by the probability $(1 - q)$:

$$T^2_{\text{lost}} = (1 - q)\frac{1}{T} \int_{C_p}^{T} (p(t + D + R) + (1 - p)0)\, dt$$

(3) **Predictions taken into account:** We now compute the overhead due to a prediction which we trust (hence we checkpoint just before its date). If the prediction is an actual fault, we lose $C_p + D + R$ seconds, but if it is not, we lose the unnecessary extra checkpoint time C_p. The expected time lost is now weighted by the probability q and becomes

$$T^3_{\text{lost}} = q\frac{1}{T} \int_{C_p}^{T} (p(C_p + D + R) + (1 - p)C_p)\, dt$$

We derive the final value of $\text{WASTE}_{\text{fault}}$:

$$\text{WASTE}_{\text{fault}} = \frac{1}{\mu_{\text{NP}}} \left[\frac{T}{2} + D + R\right] + \frac{1}{\mu_{\text{P}}} \left[T^1_{\text{lost}} + T^2_{\text{lost}} + T^3_{\text{lost}}\right]$$

This final expression comes from the disjunction of all possibles cases, using the Law of Total Probability [89, p.23]. The waste comes either from non-predicted faults or from predictions. In the latter case, we have analyzed the three possible sub-cases and weighted them with their respective probabilities. After simplifications, we obtain

$$\text{WASTE}_{\text{fault}} = \frac{1}{\mu} \left((1 - rq)\frac{T}{2} + D + R + \frac{qr}{p} C_p - \frac{qrC_p^2}{pT}(1 - p/2) \right) \tag{2.2}$$

We could now plug this expression back into Equation (2.1) to compute the value of T that minimizes the total waste. Instead, we move on to describing the refined algorithm, and we minimize the waste for the refined strategy, since it always induces a smaller waste.

2.3.2 Refined policy

We refine now the approach and consider different trust strategies, depending upon the time in the period where the prediction takes place. Intuitively, the later this happens in the period, the more likely we are inclined to trust the predictor, because the amount of work that we could lose gets larger and larger. As before, we cannot take into account a fault predicted to happen less than C_p units of time after the beginning of the period. Therefore, we focus on what happens in the period after time C_p. Formally, we divide the interval $[C_p, T]$ into n sub-intervals $[\beta_i; \beta_{i+1}]$ for $i \in \{0, \cdots, n - 1\}$, where $\beta_0 = C_p$ and $\beta_n = T$. For each sub-interval $[\beta_i; \beta_{i+1}]$, we trust the predictor with probability q_i. We aim at determining the values of n, β_i, and q_i that minimize the waste. As mentioned before, intuition tells us that the q_i values should be non-decreasing. We prove below a somewhat unexpected theorem. In the

optimal strategy, there is either one or two different q_i values, and these values are 0 or 1. This means that we should *never* trust the predictor in the beginning of a period, and always trust it in the end of the period.

We formally express this striking result below. Let $\beta_{\text{lim}} = \frac{C_p}{p}$. The optimal strategy is provided by Theorem 2.1 below. We first prove the following proposition:

Proposition 2.1. *The values of β_i and q_i that minimize the waste satisfy the following conditions:*
(i) For all i such that $\beta_{i+1} \leq \beta_{lim}$, $q_i = 0$.
(ii) For all i such that $\beta_i \geq \beta_{lim}$, $q_i = 1$.

Proof. First we compute the waste with the refined algorithm, using Equation (2.1). The formula for $\text{WASTE}_{\text{fault}}$ is similar to Equation (2.2) on each interval:

$$\begin{aligned}\text{WASTE} = \frac{C}{T} + \left(1 - \frac{C}{T}\right)\left[\frac{1}{\mu_{\text{NP}}}\left(\frac{T}{2} + D + R\right)\right. \\ + \frac{1}{\mu_{\text{P}}}\sum_{i=0}^{n-1}\left(q_i \int_{\beta_i}^{\beta_{i+1}} \frac{(p(C_p + D + R) + (1-p)C_p)}{T} dt\right. \\ \left.\left. + \ (1 - q_i)\int_{\beta_i}^{\beta_{i+1}} \frac{p(t + D + R)}{T} dt\right)\right].\end{aligned}$$

Now, consider a fixed value of i and express the value of WASTE as a function of q_i:

$$\text{WASTE} = K + \left(1 - \frac{C}{T}\right)\frac{q_i}{\mu_{\text{P}}}\int_{\beta_i}^{\beta_{i+1}}\left(\frac{C_p}{T} - \frac{pt}{T}\right)dt$$

where K does not depend on q_i. From the sign of the function to be integrated, one sees that WASTE is minimized when $q_i = 0$ if $\beta_{i+1} \leq \beta_{\text{lim}} = \frac{C_p}{p}$, and when $q_i = 1$ if $\beta_i \geq \beta_{\text{lim}}$. ■

Theorem 2.1. *The optimal algorithm takes proactive actions if and only if the prediction falls in the interval $[\beta_{lim}, T]$.*

Proof. From Proposition 2.1, the values for q_i are optimally defined for every i but one: we do not know the optimal value if there exists i_0 such that $\beta_{i_0} < \beta_{\text{lim}} < \beta_{i_0+1}$. Then let us consider the waste where q_{i_0} is replaced by $q_{i_0}^{(1)}$ on $[\beta_{i_0}, \beta_{\text{lim}}]$ and by $q_{i_0}^{(2)}$ on $[\beta_{\text{lim}}, \beta_{i_0+1}]$. The new waste is necessarily smaller than the one with only q_{i_0}, since we relaxed the constraint. We know from Proposition 2.1 that the optimal solution is then to have $q_{i_0}^{(1)} = 0$ and $q_{i_0}^{(2)} = 1$. ■

Let us now compute the value of the waste with the optimal algorithm. There are two cases, depending upon whether $T \leq \beta_{\text{lim}}$ or not. For values of T smaller than β_{lim}, Theorem 2.1 shows that the optimal algorithm never takes any proactive action; in that case the waste is given by Equation (1.14) in Chapter 1. For values of T larger than $\beta_{\text{lim}} = \frac{C_p}{p}$, we compute the waste due to predictions as

$$\begin{aligned}&\frac{1}{\mu_{\text{P}}}\frac{1}{T}\left(\int_0^{C_p/p} p(t + D + R)dt + \int_{C_p/p}^{T}(p(C_p + D + R) + (1-p)C_p)dt\right) \\ &= \frac{r}{p\mu}\left(p(D + R) + C_p - \frac{C_p^2}{2pT}\right).\end{aligned}$$

Indeed, in accordance with Theorem 2.1, no prediction is taken into account in the interval $[0, \frac{C_p}{p}]$, while all predictions are taken into account in the interval $[\frac{C_p}{p}, T]$. Adding the waste due to unpredicted faults, namely $\frac{1}{\mu_{\text{NP}}}\left[\frac{T}{2} + D + R\right]$, we derive

$$\text{WASTE}_{\text{fault}} = \frac{1}{\mu}\left((1-r)\frac{T}{2} + \frac{r}{p}C_p\left(1 - \frac{1}{2p}\frac{C_p}{T}\right) + D + R\right).$$

Plugging this value into Equation (2.1), we obtain the total waste when $\frac{C_p}{p} \leq T$:

$$\begin{aligned}\text{WASTE} &= \frac{C}{T} + \frac{1}{\mu}\left((1-r)\frac{T}{2} + \frac{r}{p}C_p\left(1 - \frac{1}{2p}\frac{C_p}{T}\right) + D + R\right)\left(1 - \frac{C}{T}\right) \\ &= \frac{rCC_p^2}{2p^2}\frac{1}{\mu T^2} + \left(\mu C - \frac{rC_p^2}{2p^2} - C\left(\frac{rC_p}{p} + D + R\right)\right)\frac{1}{\mu T} + \frac{1-r}{2\mu}T \\ &\quad + \frac{-(1-r)\frac{C}{2} + \frac{rC_p}{p} + D + R}{\mu}.\end{aligned}$$

Altogether, the expression for the total waste becomes:

$$\begin{cases} \text{WASTE}_1(T) = \frac{C\left(1-\frac{D+R}{\mu}\right)}{T} + \frac{D+R-C/2}{\mu} + \frac{1}{2\mu}T & \text{if } \frac{C_p}{p} \geq T \\ \\ \text{WASTE}_2(T) = \frac{rCC_p^2}{2\mu p^2}\frac{1}{T^2} + \frac{\left(C\left(1-\frac{\frac{rC_p}{p}+D+R}{\mu}\right) - \frac{rC_p^2}{2\mu p^2}\right)}{T} + \frac{-(1-r)\frac{C}{2} + \frac{rC_p}{p} + D + R}{\mu} + \frac{1-r}{2\mu}T & \text{if } \frac{C_p}{p} \leq T. \end{cases} \tag{2.3}$$

One can check that when $r = 0$ (no error predicted, hence no proactive action in the algorithm), then WASTE_1 and WASTE_2 coincide. We also check that both values coincide for $T = \frac{C_p}{p}$. We show how to minimize the waste in Equation (2.3) in Section 2.3.3.

2.3.3 Waste minimization

In this section we focus on minimizing the waste in Equation (2.3). Recall from Section 1.1.1 that, by construction, we always have to enforce the constraint $T \geq C$. First, consider the case where $C \leq \frac{C_p}{p}$. On the interval $T \in [C, \frac{C_p}{p}]$, we retrieve the optimal value found in Chapter 1, and derive that WASTE_1, the waste when predictions are not taken into account, is minimized for

$$T_{\text{NoPred}} = \max\left(C, \min\left(T_{\text{RFO}}, \frac{C_p}{p}\right)\right). \tag{2.4}$$

Indeed, the optimal value should belong to the interval $[C, \frac{C_p}{p}]$, and the function WASTE_1 is convex. If the extremal solution $\sqrt{2(\mu - (D+R))C}$ does not belong to this interval, then the optimal value is one of the bounds of the interval.

On the interval $T \in \left[\frac{C_p}{p}, +\infty\right)$, we find the optimal solution by differentiating twice WASTE_2 with respect to T. Writing $\text{WASTE}_2(T) = \frac{u}{T^2} + \frac{v}{T} + w + xT$ for simplicity, we obtain $\text{WASTE}_2''(T) = \frac{2}{T^3}\left(\frac{3u}{T} + v\right)$. Here, a key parameter is the sign of :

$$v = \left(C\left(1 - \frac{\frac{rC_p}{p} + D + R}{\mu}\right) - \frac{rC_p^2}{2\mu p^2}\right)$$

We detail the case $v \geq 0$ in the following, because it is the most frequent with realistic parameter sets; we do have $v \geq 0$ for all the whole range of simulations in Section 2.4. For the sake of completeness, we will briefly discuss the case $v < 0$ in the comments below.

When $v \geq 0$, we have $\text{WASTE}_2''(T) \geq 0$, so that WASTE_2 is convex on the interval $\left[\frac{C_p}{p}, +\infty\right)$ and admits a unique minimum T_{extr}. Note that T_{extr} can be computed either numerically or using Cardano's method, since it is the unique real root of a polynomial of degree 3. The optimal solution on $\left[\frac{C_p}{p}, +\infty\right)$ is then: $T_{\text{PRED}} = \max\left(T_{\text{extr}}, \frac{C_p}{p}\right)$.

It remains to consider the case where $\frac{C_p}{p} < C$. In fact, it suffices to add the constraint that the value of T_{PRED} should be greater than C, that is:

$$T_{\text{PRED}} = \max\left(C, \max\left(T_{\text{extr}}, \frac{C_p}{p}\right)\right) \tag{2.5}$$

Finally, the optimal solution for the waste is given by the minimum of the following two values:

$$\begin{cases} \frac{C\left(1-\frac{D+R}{\mu}\right)}{T_{\text{NOPRED}}} + \frac{D+R-C/2}{\mu} + \frac{1}{2\mu}T_{\text{NOPRED}} \\ \\ \frac{rCC_p^2}{2\mu p^2}\frac{1}{T_{\text{PRED}}^2} + \frac{\left(C\left(1-\frac{\frac{rC_p}{p}+D+R}{\mu}\right)-\frac{rC_p^2}{2\mu p^2}\right)}{T_{\text{PRED}}} + \frac{-(1-r)\frac{C}{2}+\frac{rC_p}{p}+D+R}{\mu} + \frac{1-r}{2\mu}T_{\text{PRED}} \end{cases}$$

We make a few observations:

- Just as for Equation (1.15) in Chapter 1, mathematical rigor calls for capping the values of D, R, C, C_p and T in front of the MTBF. The only difference is that we should replace μ by μ_e: this is to account for the occurrence rate of all events, be they unpredicted faults or predictions.
- While the expression of the waste looks complicated, the numerical value of the optimal period can easily be computed in all cases. We have dealt with the case $v \geq 0$, where v is the coefficient of $1/T$ in $\text{WASTE}_2(T) = \frac{u}{T^2} + \frac{v}{T} + w + xT$. When $v < 0$ we only needs to compute all the nonnegative real roots of a polynomial of degree 3, and check which one leads to the best value. More precisely, these root(s) partition the admissible interval $\left[\frac{C_p}{p}, +\infty\right)$ into several sub-intervals, and the optimal value is either a root or a sub-interval bound.
- In many practical situations, when μ is large enough, we can dramatically simplify the expression of $\text{WASTE}_2(T)$: we have $T = O(\sqrt{\mu})$, the term $\frac{u}{T^2}$ becomes negligible, checkpoint parameters become negligible in front of μ, and we derive the approximated value $\sqrt{\frac{2\mu C}{1-r}}$. This value can be seen as an extension of Equation (1.15) giving T_{RFO}, where μ is replaced by $\frac{\mu}{1-r}$: faults are replaced by non-predicted faults, and the overhead due to false predictions is negligible. As a word of caution, recall that this conclusion is valid only when μ is very large in front of all other parameters.

2.4 Simulation results

We start by presenting the simulation framework (Section 2.4.1). Then we report results using synthetic traces (Section 2.4.2) and log-based traces (Section 2.4.3). Finally, we assess the respective impact of the two key parameters of a predictor, its recall and its precision, on checkpointing strategies (Section 2.4.4).

2.4.1 Simulation framework

Scenario generation – In order to check the accuracy of our model and of our analysis, and to assess the potential benefits of predictors, we study the performance of our new solutions and of pre-existing ones using a discrete-event simulator. The simulation engine generates a random trace of faults. Given a set of p processors, a failure trace is a set of failure dates for each processor over a fixed time horizon h (set to 2 years). Given the distribution of inter-arrival times at a processor, for each processor we generate a trace via independent sampling until the target time horizon is reached. The job start time is assumed to be one-year to avoid side-effects related to the synchronous initialization of all nodes/processors. We consider two types of failure traces, namely synthetic and log-based.

Synthetic failure traces – The simulation engine generates a random trace of faults parameterized either by an Exponential fault distribution or by Weibull distribution laws with shape parameter either 0.5 or 0.7. Note that Exponential faults are widely used for theoretical studies, while Weibull faults are representative of the behavior of real-world platforms [58, 110, 81, 59]. For example, Heien et al. [59] have studied the failure distribution for 6 sources of failures (storage devices, NFS, batch system, memory and processor cache errors, etc.), and the aggregate failure distribution. They have shown that the aggregate failure distribution is best modeled by a Weibull distribution with a shape parameter that is between 0.5841 and 0.7097.

The Jaguar platform, which comprised $N = 45,208$ processors, is reported to have experienced about one fault per day [128], which leads to an individual (processor) MTBF μ_{ind} equal to $\frac{45,208}{365} \approx 125$ years. Therefore, we set the individual (processor) MTBF to $\mu_{\text{ind}} = 125$ years. We let the total number of processors N vary from $N = 16,384$ to $N = 524,288$, so that the platform MTBF μ varies from $\mu = 4,010$ min (about 2.8 days) down to $\mu = 125$ min (about 2 hours). Whatever the underlying failure distribution, it is scaled so that its expectation corresponds to the platform MTBF μ. The application size is set to $\text{TIME}_{\text{base}} = 10,000$ years/N.

Log-based failure traces – To corroborate the results obtained with synthetic failure traces, and to further assess the performance of our algorithms, we also perform simulations using the failure logs of two production clusters. We use logs of the largest clusters among the preprocessed logs in the *Failure trace archive* [73], i.e., for clusters at the Los Alamos National Laboratory [110]. In these logs, each failure is tagged by the node —and not the processor— on which the failure occurred. Among the 26 possible clusters, we opted for the logs of the only two clusters with more than 1,000 nodes. The motivation is that we need a sample history sufficiently large to simulate platforms with more than ten thousand nodes. The two chosen logs are for clusters 18 (LANL18) and 19 (LANL19) in the archive (referred to as 7 and 8 in the description of the clusters [110]). For each log, we record the set $\mathcal{S}$ of availability intervals. The discrete failure distribution for the simulation is generated as follows: the conditional probability $\mathbb{P}(X \geq t \mid X \geq \tau)$ that a node stays up for a duration t, knowing that it has been up for a duration τ, is set to the ratio of the number of availability durations in $\mathcal{S}$ greater than or equal to t, over the number of availability durations in $\mathcal{S}$ greater than or equal to τ.

The two clusters used for computing our log-based failure distributions consist of 4-processor nodes. Hence, to simulate a platform of, say, 2^{16} processors, we generate 2^{14} failure traces, one for each 4-processor node. In the logs the individual (processor) MTBF is $\mu_{\text{ind}} = 691$ days for the LANL18 cluster, and $\mu_{\text{ind}} = 679$ days for the LANL19 cluster. The LANL18 and LANL19 traces are logs for systems which comprised 4,096 processors. Using these logs to generate traces for a system made of $524,288$ processors, as the largest platforms we consider with synthetic failure traces, would lead to an obvious risk of oversampling. Therefore, we limit the size of the log-based traces we generate: we let the total number of processors N varies from $N = 1,024$ to $N = 131,072$, so that the platform MTBF μ varies from $\mu = 971$ min (about 16 hours) down to $\mu = 7.5$ min. The application size is set to

$\text{TIME}_{\text{base}} = 250$ years/N.

Predicted failures and false predictions – Once we have generated a failure trace, we need to determine which faults are predicted and which are not. In order to do so, we consider all faults in a trace one by one. For each of them, we randomly decide, with probability r, whether it is predicted.

We use the simulation engine to generate a random trace of false predictions. The main problem is to decide the shape of the distribution that false predictions should follow. To the best of our knowledge, no published study ever addressed that problem. For synthetic failure traces, we report results when false predictions follow the same distribution than faults (except, of course, that both distributions do not have the same mean value). Results are quite similar when false predictions are generated according to a uniform distribution. For log-based failures, we only report results when false predictions are generated according to a uniform distribution (because we believe that scaling down a discrete, actual distribution may not be meaningful).

The distribution of false predictions is always scaled so that its expectation is equal to $\frac{\mu_P}{1-p} = \frac{p\mu}{r(1-p)}$, the inter-arrival time of false predictions. Finally, the failure trace and the false-prediction trace are merged to produce the final trace including all events (true predictions, false predictions, and non predicted faults). Each reported value is the average over 100 randomly generated instances.

Checkpointing, recovery, and downtime costs – The experiments use parameters that are representative of current and forthcoming large-scale platforms [25, 43]. We take $C = R = 10$ min, and $D = 1$ min for the synthetic failure traces. For the log-based traces we consider smaller platforms. Therefore, we take $C = R = 1$ min, and $D = 6$s. Whatever the trace, we consider three scenarios for the proactive checkpoints: either proactive checkpoints are (i) exactly as expensive as periodic ones ($C_p = C$), (ii) ten times cheaper ($C_p = 0.1C$), and (iii) two times more expensive ($C_p = 2C$).

Heuristics – In the simulations, we compare four checkpointing strategies:

- RFO is the checkpointing strategy of period $T = \sqrt{2(\mu - (D + R))C}$ (see Chapter 1).
- OPTIMALPREDICTION is the refined algorithm described in Section 2.3.2.
- To assess the quality of each strategy, we compare it with its BESTPERIOD counterpart, defined as the same strategy but using the best possible period T. This latter period is computed via a brute-force numerical search for the optimal period (each tested period is evaluated on 100 randomly generated traces, and the period achieving the best average performance is elected as the "best period").

Fault predictors – We experiment using the characteristics of two predictors from the literature: one accurate predictor with high recall and precision [125], namely with $p = 0.82$ and $r = 0.85$, and another predictor with intermediate recall and precision [131], namely with $p = 0.4$ and $r = 0.7$.

In practice, a predictor will not be able to predict the exact time at which a predicted fault will strike the system. Therefore, in the simulations, when a predictor predicts that a failure will strike the system at a date t (true prediction), the failure actually occurs exactly at time t for heuristic OPTIMALPREDICTION, and between time t and time $t + 2C$ for heuristic INEXACTPREDICTION (the probability of fault is uniformly distributed in the time-interval). OPTIMALPREDICTION can thus be seen as a best case. The comparison between OPTIMALPREDICTION and INEXACTPREDICTION enables us to assess the impact of the time imprecision of predictions, and to show that the obtained results are quite robust to this type of imprecision. The choice of an interval length of $2C$ is quite arbitrary. For synthetic traces, this corresponds to 1,200 s, which is quite a significant imprecision.

2.4.2 Simulations with synthetic traces

Figures 2.2 and 2.3 show the average waste degradation for the two checkpointing policies, and for their BESTPERIOD counterparts, for both predictors. The waste is reported as a function of the number of

processors N. We draw the plots as a function of the number of processors N rather than of the platform MTBF $\mu = \mu_{\text{ind}}/N$, because it is more natural to see the waste increase with larger platforms. However, recall that this work is agnostic of the granularity of the processing elements and intrinsically focuses on the impact of the MTBF on the waste.

We also report job execution times, in Table 2.2 when fault distribution follows an Exponential distribution law, and in Tables 2.3 and 2.4 for a Weibull distribution law with shape parameter $k = 0.7$ and $k = 0.5$ respectively.

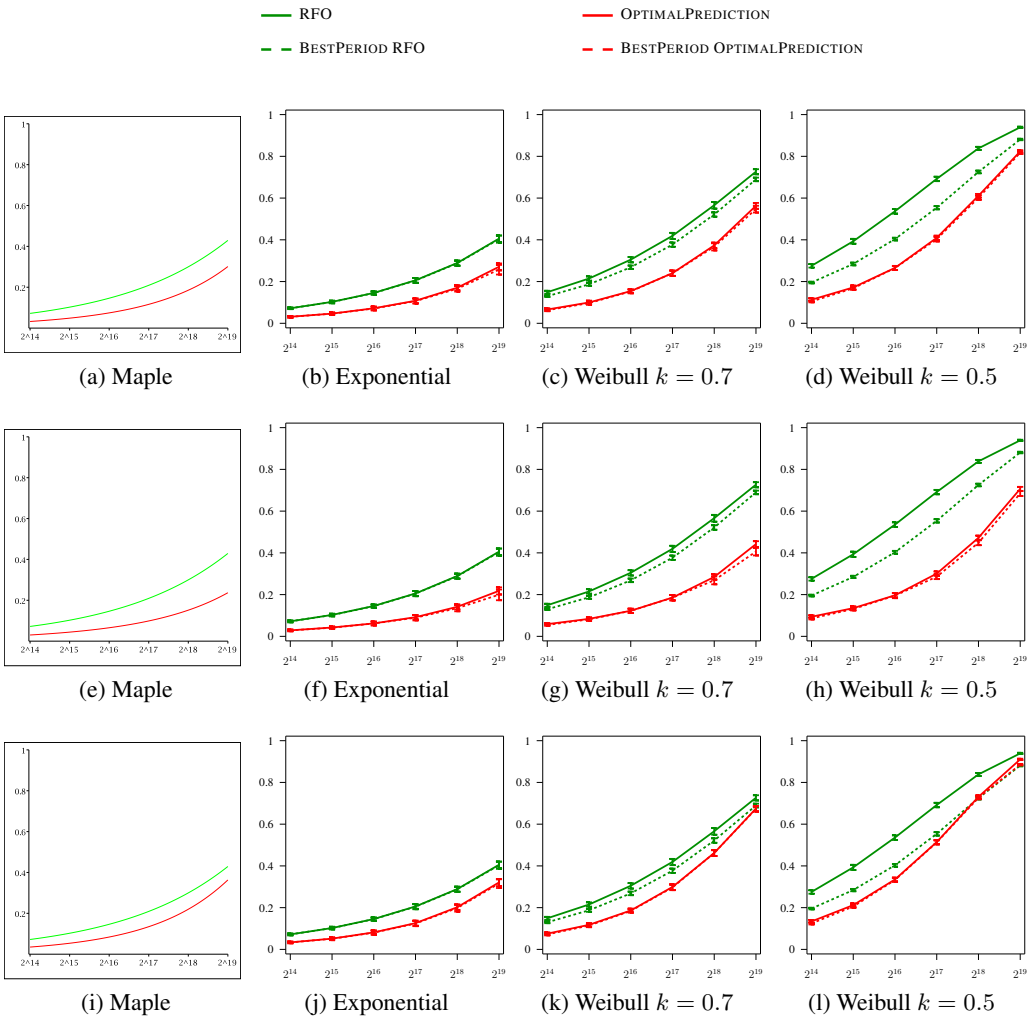
Validation of the theoretical study – We used Maple to analytically compute and plot the optimal value of the waste for both the algorithm taking predictions into account, OPTIMALPREDICTION, and for the algorithm ignoring them, RFO. In order to check the accuracy of our model, we have compared these results with results obtained with the discrete-event simulator.

We first observe that there is a very good correspondence between analytical results and simulations in Figures 2.2 and 2.3. In particular, the Maple plots and the simulations for Exponentially distributed faults are very similar. This shows the validity of the model and of its analysis. Another striking result is that OPTIMALPREDICTION has the same waste as its BESTPERIOD counterpart, even for Weibull fault distributions, in all but the most extreme cases. In the other cases, the waste achieved by OPTIMALPREDICTION is very close to that of its BESTPERIOD counterpart. This demonstrates the very good quality of our checkpointing period T_{PRED}. These conclusions are valid regardless of the cost ratio of periodic and proactive checkpoints.

In Tables 2.2 through 2.4 we report the execution times obtained when using the expression of T given by Young [124] and Daly [35] (denoted respectively as YOUNG and DALY) to assess whether T_{RFO} is a better approximation. (Recall that these three approaches ignore the predictions, which explains why the numbers are identical on both sides of each table.) The expressions of T given by YOUNG, DALY, and RFO are identical for Exponential distributions and the three heuristics achieve the same performance (Table 2.2). This confirms the analytical evaluation of Table 1.1 in Chapter 1. For Weibull distributions (Tables 2.3 and 2.4), RFO achieves lower makespan, and the difference becomes even more significant as the size of the platform increases. Moreover, it is striking to observe in Table 2.4 that job execution time increases together with the number for processors (from $N = 2^{16}$ to $N = 2^{19}$) if the checkpointing period is DALY or YOUNG. On the contrary, job execution time (rightfully) decreases when using RFO, even if the decrease is moderate with respect to the increase of the platform size. Altogether, the main (striking) conclusion is that RFO should be preferred to both classical approaches for Weibull distributions.

The benefits of prediction – The second observation is that the prediction is useful for the vast majority of the set of parameters under study! In addition, when proactive checkpoints are cheaper than periodic ones, the benefits of fault prediction are increased. On the contrary, when proactive checkpoints are more expensive than periodic ones, the benefits of fault prediction are greatly reduced. One can even observe that the waste with prediction is not better than without prediction in the following scenario: $C_p = 2C$, and using the limited-quality predictor ($p = 0.4$, $r = 0.7$) with 2^{19} processors, see Figures 2.3(i),(j),(k), and (l).

In Tables 2.2 through 2.4 we compute the gain (expressed in percentage) achieved by OPTIMALPREDICTION over RFO. As a general trend, we observe that the gains due to predictions are more important when the distribution law is further apart from an Exponential distribution. Indeed, the largest gains are when the fault distribution follows a Weibull law of parameter 0.5. Using OPTIMALPREDICTION in conjunction with a "good" fault predictor we report gains up to 66% when there is a large number of processors (2^{19}). The gain is still of 37% with 2^{16} processors. Using a predictor with limited recall and precision, OPTIMALPREDICTION can still decrease the execution time by 47% with 2^{19} processors, and 31% with 2^{16} processors. In all tested cases, the decrease of the execution times is significant. Gains are

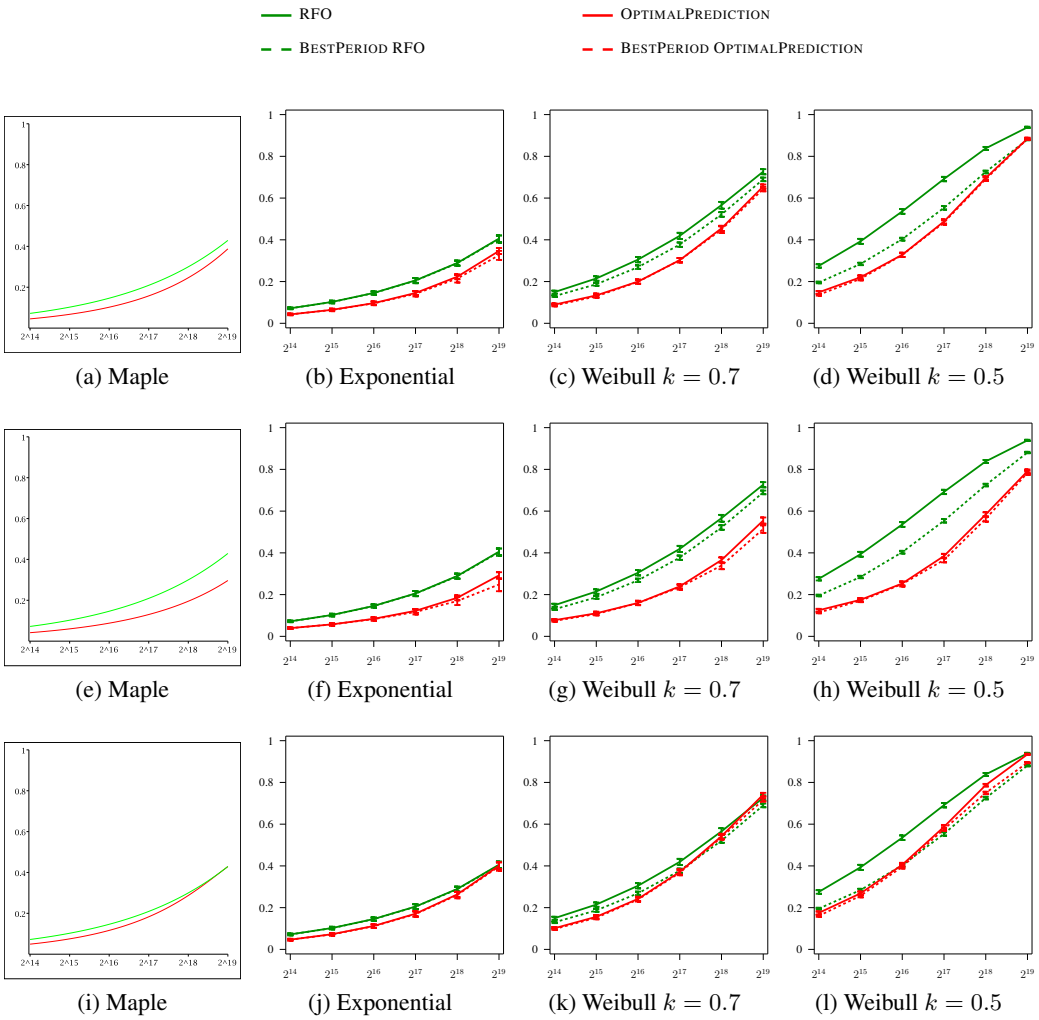


(a) Maple (b) Exponential (c) Weibull $k = 0.7$ (d) Weibull $k = 0.5$

(e) Maple (f) Exponential (g) Weibull $k = 0.7$ (h) Weibull $k = 0.5$

(i) Maple (j) Exponential (k) Weibull $k = 0.7$ (l) Weibull $k = 0.5$

Figure 2.2: Waste (y-axis) for the different heuristics as a function of the platform size (x-axis), with $p = 0.82$, $r = 0.85$, $C_p = C$ (first row), $C_p = 0.1C$ (second row), or $C_p = 2C$ (third row) and with a trace of false predictions parameterized by a distribution identical to the distribution of the failure trace.



(a) Maple (b) Exponential (c) Weibull $k = 0.7$ (d) Weibull $k = 0.5$

(e) Maple (f) Exponential (g) Weibull $k = 0.7$ (h) Weibull $k = 0.5$

(i) Maple (j) Exponential (k) Weibull $k = 0.7$ (l) Weibull $k = 0.5$

Figure 2.3: Waste (y-axis) for the different heuristics as a function of the platform size (x-axis), with $p = 0.4$, $r = 0.7$, $C_p = C$ (first row), $C_p = 0.1C$ (second row), or $C_p = 2C$ (third row) and with a trace of false predictions parameterized by a distribution identical to the distribution of the failure trace.

less important with Weibull laws of shape parameter $k = 0.7$, however they are still reaching a minimum of 13% with 2^{16} processors, and up to 38% with 2^{19} processors. Finally, gains are further reduced with an Exponential law. They are still reaching at least 5% with 2^{16} processors, and up to 19% with 2^{19} processors.

The performance of INEXACTPREDICTION shows that using a fault predictor remains largely beneficial even in the presence of large uncertainties on the time the predicted faults will actually occur (see Tables 2.2, 2.3, and 2.4). When $N = 2^{16}$ the degradation with respect to OPTIMALPREDICTION is of 3% for a Weibull law with shape parameter $k = 0.7$, and the minimum gain over RFO is still of 10%. When the shape parameter of the Weibull law is $k = 0.5$, the degradation is of 7% when, for a minimum gain of 26% over RFO.

$C_p = C$	Execution time (in days) ($p = 0.82$, $r = 0.85$)		Execution time (in days) ($p = 0.4$, $r = 0.7$)	
	2^{16} procs	2^{19} procs	2^{16} procs	2^{19} procs
YOUNG	65.2	11.7	65.2	11.7
DALY	65.2	11.8	65.2	11.8
RFO	65.2	11.7	65.2	11.7
OPTIMALPREDICTION	60.0 (8%)	9.5 (19%)	61.7 (5%)	10.7 (8%)
INEXACTPREDICTION	60.6 (7%)	10.2 (13%)	62.3 (4%)	11.4 (3%)

Table 2.2: Job execution times for an Exponential distribution, and gains due to the fault predictor (with respect to the performance of RFO).

$C_p = C$	Execution time (in days) ($p = 0.82$, $r = 0.85$)		Execution time (in days) ($p = 0.4$, $r = 0.7$)	
	2^{16} procs	2^{19} procs	2^{16} procs	2^{19} procs
YOUNG	81.3	30.1	81.3	30.1
DALY	81.4	31.0	81.4	31.0
RFO	80.3	25.5	80.3	25.5
OPTIMALPREDICTION	65.9 (18%)	15.9 (38%)	69.7 (13%)	20.2 (21%)
INEXACTPREDICTION	68.0 (15%)	20.3 (20%)	72.0 (10%)	24.6 (4%)

Table 2.3: Job execution times for a Weibull distribution with shape parameter $k = 0.7$, and gains due to the fault predictor (with respect to the performance of RFO).

$C_p = C$	Execution time (in days) ($p = 0.82$, $r = 0.85$)		Execution time (in days) ($p = 0.4$, $r = 0.7$)	
	2^{16} procs	2^{19} procs	2^{16} procs	2^{19} procs
YOUNG	125.5	171.8	125.5	171.8
DALY	125.8	184.7	125.8	184.7
RFO	120.2	114.8	120.2	114.8
OPTIMALPREDICTION	75.9 (37%)	39.5 (66%)	83.0 (31%)	60.8 (47%)
INEXACTPREDICTION	82.0 (32%)	60.8 (47%)	89.4 (26%)	76.6 (33%)

Table 2.4: Job execution times for a Weibull distribution with shape parameter $k = 0.5$, and gains due to the fault predictor (with respect to the performance of RFO).

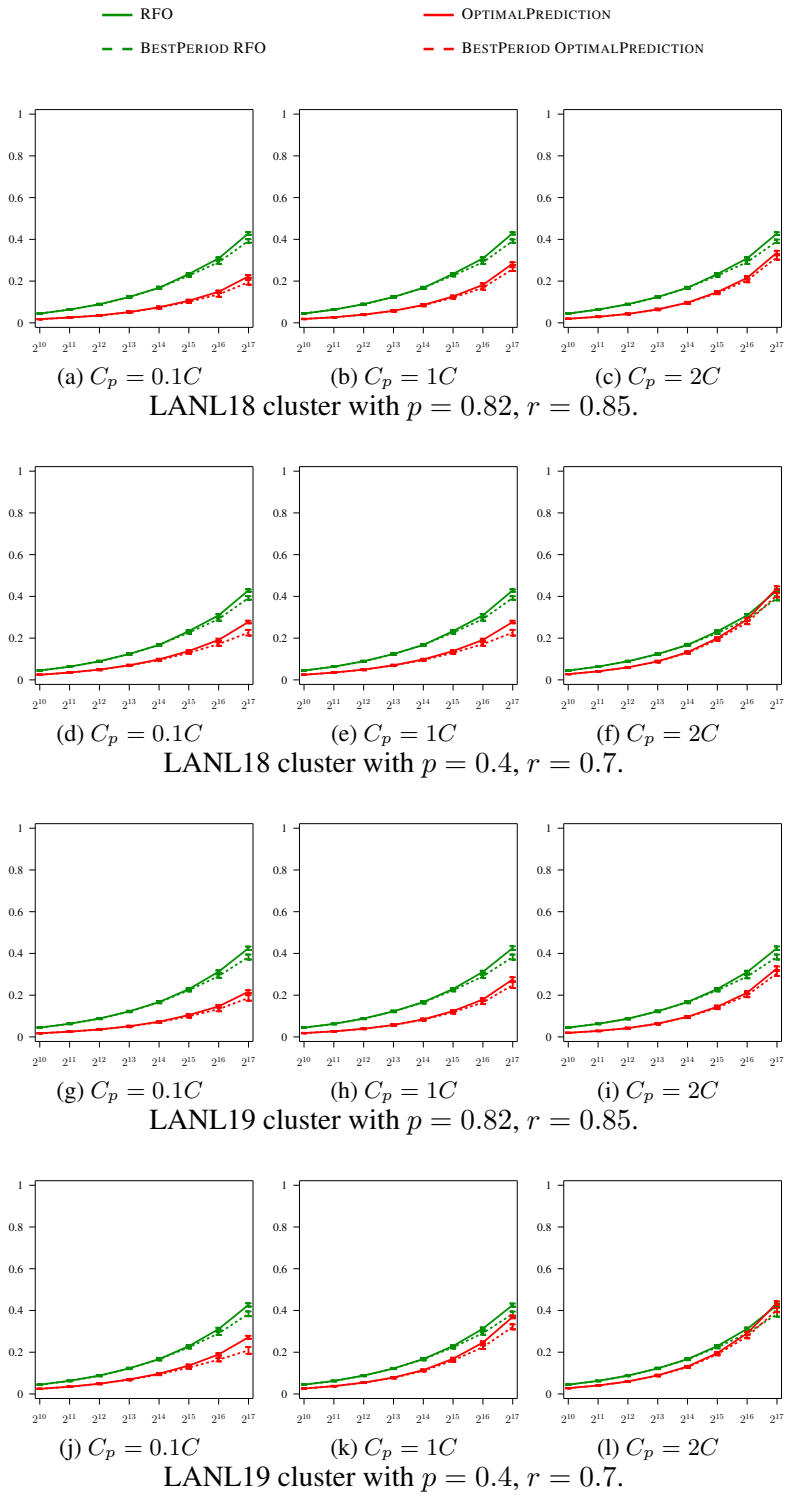
2.4.3 Simulations with log-based traces

Figure 2.4 shows the average waste degradation for the two checkpointing policies, and for their BESTPERIOD counterparts, for both predictors, both traces, and the three scenarios for proactive checkpoints. Tables 2.5 and 2.6 present job execution times for RFO, OPTIMALPREDICTION, and INEXACTPREDICTION, for both traces and for platform sizes smaller than as the ones reported in Tables 2.2 through 2.4 for synthetic traces. The waste for RFO is closer to its BESTPERIOD counterpart with log-based traces than with Weibull-based traces. As a consequence, when prediction with OPTIMALPREDICTION is beneficial, it is beneficial with respect to both RFO, and to RFO's BESTPERIOD.

Overall, we observe similar results and reach the same conclusions with log-based traces as with synthetic ones. The waste of OPTIMALPREDICTION is very close to that of its BESTPERIOD counterpart for platforms containing up to 2^{16} processors. This demonstrates the validity of our analysis for the actual traces considered. The waste of OPTIMALPREDICTION is often significantly larger than that of its BESTPERIOD counterpart for platforms containing 2^{17} processors. The problem with the largest considered platforms may be due to oversampling. Indeed, the original logs recorded events for platforms comprising only 4,096 processors and respectively contained only 3,010 and 2,343 availability intervals.

As with synthetic failure traces, prediction turns out to be useful for the vast majority of tested configurations. The only cases when prediction is not useful is with the "bad" predictor ($r = 0.7$ and $p = 0.4$), when the cost of proactive checkpoint is larger than the cost of periodic checkpoints ($C_p = 2C$), and when considering the largest of platforms ($N = 2^{17}$). This extreme case is, however, the only one for which prediction is not beneficial. It is not surprising that predictions are not useful when there are a lot of false predictions that require the use of expensive proactive actions. Looking at Tables 2.5 and 2.6, one could remark that performance gains due to the predictions are similar to the ones observed with Exponential-based traces, and are significantly smaller than the ones observed with Weibull-based traces. However, recall that we remarked that gains increase with the size of the platform, and that we consider smaller platforms when using log-based traces.

Finally, the imprecision related to the time where predicted faults strike, induces a performance degradation. However, this degradation is rather limited for the most efficient of the two predictors considered, or when the platform size is not too large.



(a) $C_p = 0.1C$ (b) $C_p = 1C$ (c) $C_p = 2C$

LANL18 cluster with $p = 0.82$, $r = 0.85$.

(d) $C_p = 0.1C$ (e) $C_p = 1C$ (f) $C_p = 2C$

LANL18 cluster with $p = 0.4$, $r = 0.7$.

(g) $C_p = 0.1C$ (h) $C_p = 1C$ (i) $C_p = 2C$

LANL19 cluster with $p = 0.82$, $r = 0.85$.

(j) $C_p = 0.1C$ (k) $C_p = 1C$ (l) $C_p = 2C$

LANL19 cluster with $p = 0.4$, $r = 0.7$.

Figure 2.4: Waste (y-axis) for the different heuristics as a function of the platform size (x-axis) with failures based on the failure log of LANL clusters 18 and 19.

$C_p = C$	Execution time (in days) ($p = 0.82$, $r = 0.85$)		Execution time (in days) ($p = 0.4$, $r = 0.7$)	
	2^{14} procs	2^{17} procs	2^{14} procs	2^{17} procs
RFO	26.8	4.88	26.8	4.88
OPTIMALPREDICTION	24.4 (9%)	3.89 (20%)	25.2 (6%)	4.44 (9%)
INEXACTPREDICTION	24.7 (8%)	4.20 (14%)	25.5 (5%)	4.73 (3%)

Table 2.5: Job execution times with failures based on the failure log of LANL18 cluster, and gains due to the fault predictor (with respect to the performance of RFO).

$C_p = C$	Execution time (in days) ($p = 0.82$, $r = 0.85$)		Execution time (in days) ($p = 0.4$, $r = 0.7$)	
	2^{14} procs	2^{17} procs	2^{14} procs	2^{17} procs
RFO	26.8	4.86	26.8	4.86
OPTIMALPREDICTION	24.4 (9%)	3.85 (21%)	25.2 (6%)	4.42 (9%)
INEXACTPREDICTION	24.6 (8%)	4.14 (15%)	25.4 (5%)	4.71 (3%)

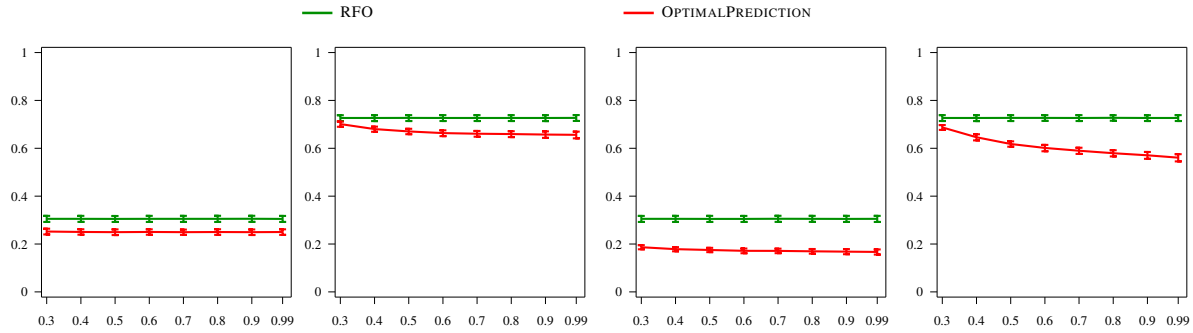
Table 2.6: Job execution times with failures based on the failure log of LANL19 cluster, and gains due to the fault predictor (with respect to the performance of RFO).

2.4.4 Recall vs. precision

In this section, we assess the impact of the two key parameters of the predictor, its recall r and its precision p. To this purpose, we conduct simulations with synthetic traces, where one parameter is fixed while the other varies. We choose two platforms, a smaller one with $N = 2^{16}$ processors (or a MTBF $\mu = 1,000\ min$) and a larger one with $N = 2^{19}$ processors (or a MTBF $\mu = 125\ min$). In both cases we study the impact of the predictor characteristics assuming a Weibull fault distribution with shape parameter either 0.5 or 0.7, under the scenario $C_p = C$.

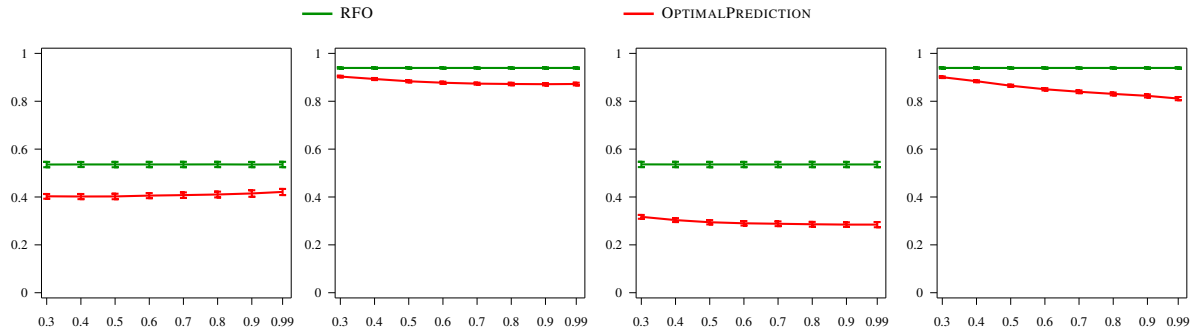
In Figures 2.5 and 2.6, we fix the value of r (either $r = 0.4$ or $r = 0.8$) and we let p vary from 0.3 to 0.99. In the four plots, we observe that the precision has a minor impact on the waste, whether it is with a Weibull distribution of shape parameter 0.7 (Figure 2.5), or a Weibull distribution of shape parameter 0.5 (Figure 2.6). In Figures 2.7 and 2.8, we conduct the converse experiment and fix the value of p (either $p = 0.4$ or $p = 0.8$), letting r vary from 0.3 to 0.99. Here we observe that increasing the recall significantly improves performance, in all but one configuration. In the configuration where improving the recall does not make a (significant) difference, there is a very large number of faults and a low precision, hence a large number of false predictions which negatively impact the performance whatever the value of the recall.

Altogether we conclude that it is more important (for the design of future predictors) to focus on improving the recall r rather than the precision p, and our results can help quantify this statement. We provide an intuitive explanation as follows: unpredicted faults prove very harmful and heavily increase the waste, while unduly checkpointing due to false predictions (usually) turns out to induce a smaller overhead.



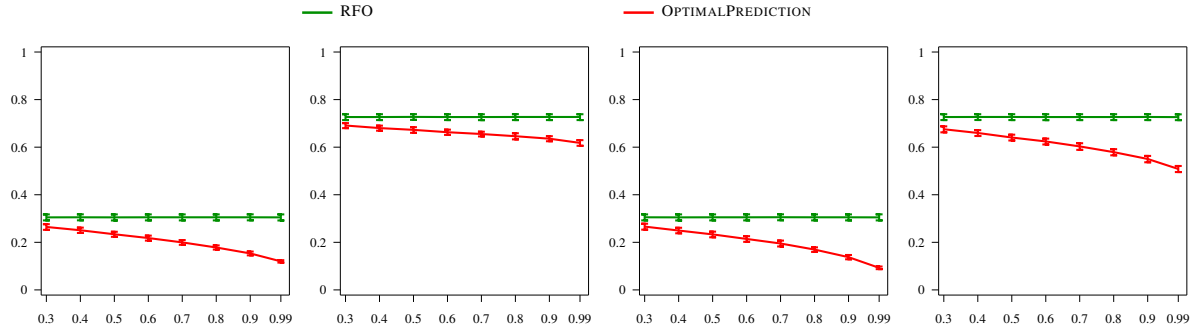
(a) $r = 0.4$, $N = 2^{16}$ (b) $r = 0.4$, $N = 2^{19}$ (c) $r = 0.8$, $N = 2^{16}$ (d) $r = 0.8$, $N = 2^{19}$

Figure 2.5: Waste (y-axis) as a function of the precision (x-axis) for a fixed recall ($r = 0.4$ and $r = 0.8$) and for a Weibull distribution of faults (with shape parameter $k = 0.7$).



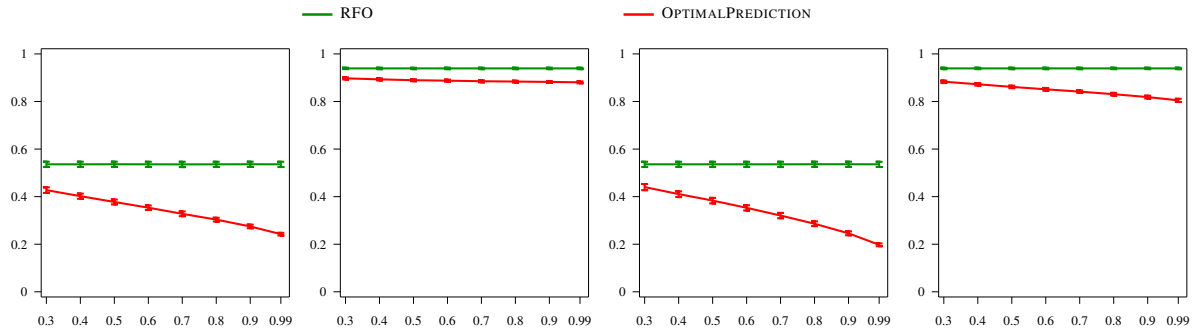
(a) $r = 0.4$, $N = 2^{16}$ (b) $r = 0.4$, $N = 2^{19}$ (c) $r = 0.8$, $N = 2^{16}$ (d) $r = 0.8$, $N = 2^{19}$

Figure 2.6: Waste (y-axis) as a function of the precision (x-axis) for a fixed recall ($r = 0.4$ and $r = 0.8$) and for a Weibull distribution of faults (with shape parameter $k = 0.5$).



(a) $p = 0.4$, $N = 2^{16}$ (b) $p = 0.4$, $N = 2^{19}$ (c) $p = 0.8$, $N = 2^{16}$ (d) $p = 0.8$, $N = 2^{19}$

Figure 2.7: Waste (y-axis) as a function of the recall (x-axis) for a fixed precision ($p = 0.4$ and $p = 0.8$) and for a Weibull distribution (k=0.7).



(a) $p = 0.4$, $N = 2^{16}$ (b) $p = 0.4$, $N = 2^{19}$ (c) $p = 0.8$, $N = 2^{16}$ (d) $p = 0.8$, $N = 2^{19}$

Figure 2.8: Waste (y-axis) as a function of the recall (x-axis) for a fixed precision ($p = 0.4$ and $p = 0.8$) and for a Weibull distribution (k=0.5).

2.5 Conclusion

In this chapter, we have studied the impact of fault prediction on periodic checkpointing. We started by extending the analysis of Section 1.1.2 to include fault predictions. We have established analytical conditions stating whether a fault prediction should be taken into account or not. More importantly, we have proven that the optimal approach is to never trust the predictor in the beginning of a regular period, and to always trust it in the end of the period; the cross-over point $\frac{C_p}{p}$ depends on the time to take a proactive checkpoint and on the precision of the predictor. This striking result is somewhat unexpected, as one might have envisioned more trust regimes, with several intermediate trust levels smoothly evolving from a "never trust" policy to an "always trust" one.

We have conducted simulations involving synthetic failure traces following either an exponential distribution law or a Weibull one. We have also used log-based failure traces. In addition, we have used exact prediction dates and uncertainty intervals for these dates. Through this extensive experiment setting, we have established the accuracy of the model, of its analysis, and of the predicted period (in the presence of a fault predictor). The simulations also show that even a not-so-good fault predictor can lead to quite a significant decrease in the application execution time. We have also shown that the most important characteristic of a fault predictor is its recall (the percentage of actually predicted faults) rather than its precision (the percentage of predictions that actually correspond to faults): *better safe than sorry*, or better prepare for a false event than miss an actual failure!

Altogether, the analytical model and the comprehensive results provided in this work enable to fully assess the impact of fault prediction on optimal checkpointing strategies. Future work will be devoted to the study of the impact of fault prediction on uncoordinated or hierarchical checkpointing protocols. Another challenging problem is to determine the best trade-off between performance and energy consumption when combining several resilience techniques such as checkpointing, prediction, and replication. We discuss this problem in Part II.

Chapter 3

Checkpointing strategies with prediction windows

3.1 Introduction

In this chapter, we refine the work on the impact of fault prediction techniques on coordinated checkpointing strategies.

Assume now that some fault prediction system is available. We remind that such a system is characterized by two critical parameters, its recall r, which is the fraction of faults that are indeed predicted, and its precision p, which is the fraction of predictions that are correct (i.e., correspond to actual faults). In the simple case where predictions are exact-date predictions, Gainaru et al. [47] and our previous work (Chapter 2) have independently shown that the optimal checkpointing period becomes $T_{\text{opt}} = \sqrt{\frac{2\mu C}{1-r}}$. This latter expression is valid only when μ is large enough. This expression can be seen as an extension of Young's formula where μ is replaced by $\frac{\mu}{1-r}$: faults are replaced by non-predicted faults, and the overhead due to false predictions is negligible.

This chapter deals with the realistic case (see [125, 79] and related work in Section 1.2) where the predictor system does not provide exact dates for predicted events, but instead provides *prediction windows*. A *prediction window* is a time interval of length I during which the predicted event is likely to happen. Intuitively, one is more at risk during such an interval than in the absence of any prediction, hence the need to checkpoint more frequently. But with which period? Should we take into account all predictions? And what is the size of the prediction window above which it proves worthwhile to use a different (smaller) checkpointing period during the prediction windows? It turns out that the answer to those questions is dramatically more complicated than when using exact-date predictions.

Main contributions. In this chapter, we extend the work of the previous chapter to a framework where the predictor predicts a fault within an interval of time I. We design several checkpointing policies that account for the different sizes of prediction windows. Then for each set of parameters, we characterize analytically the best policy. Finally, we validate the theoretical results via extensive simulations, for both Exponential and Weibull failure distributions.

The rest of the chapter is organized as follows. First we detail the framework in Section 3.2. In Section 3.3 we describe three new checkpointing policies with prediction windows, and show how to compute the optimal checkpointing periods that minimize the platform waste. Section 3.4 is devoted to

simulations. Finally, we present concluding remarks in Section 3.5.

3.2 Framework

This chapter uses a very similar model to the one introduced in Section 2.2. The only difference is the definition of the fault predictor.

A fault predictor is a mechanism that is able to predict that some faults will take place, *within some time-interval window*. Again, we assume that the predictor is able to generate its predictions early enough so that a *proactive* checkpoint can indeed be taken before or during the event. A first proactive checkpoint will typically be taken just before the beginning of the prediction window, and possibly several other ones will be taken inside the prediction window, if its size I is large enough.

As in Chapter 2, the accuracy of the fault predictor is characterized by two quantities, the *recall* and the *precision*. The recall r is the fraction of faults that are predicted while the precision p is the fraction of fault predictions that are correct.

We remind the main notations of the previous chapter that are used in this chapter in Table 3.1, updated with notations from this chapter.

p	Predictor precision: proportion of true positives among the number of predicted faults
r	Predictor recall: proportion of predicted faults among total number of faults
q	Probability to trust the predictor
MTBF	Mean Time Between Faults
N	Number of processors in the platform
μ	Platform MTBF
μ_{ind}	Individual MTBF
μ_{P}	Rate of predicted faults
μ_{NP}	Rate of unpredicted faults
μ_{e}	Rate of events (predictions or unpredicted faults)
D	Downtime
R	Recovery time
C	Duration of a regular checkpoint
C_p	Duration of a proactive checkpoint
T	Duration of a period
I	Size of the prediction window

Table 3.1: Table of main notations.

3.3 Checkpointing strategies

In this section, we introduce three new checkpointing strategies, and we determine the waste that they induce. We then proceed to computing the optimal period for each strategy.

We consider the following general scheme:

1. While no fault prediction is available, checkpoints are taken periodically with period T;
2. When a fault is predicted, we decide whether to take the prediction into account or not. This decision is randomly taken: with probability q, we trust the predictor and take the prediction into account, and, with probability $1 - q$, we ignore the prediction;

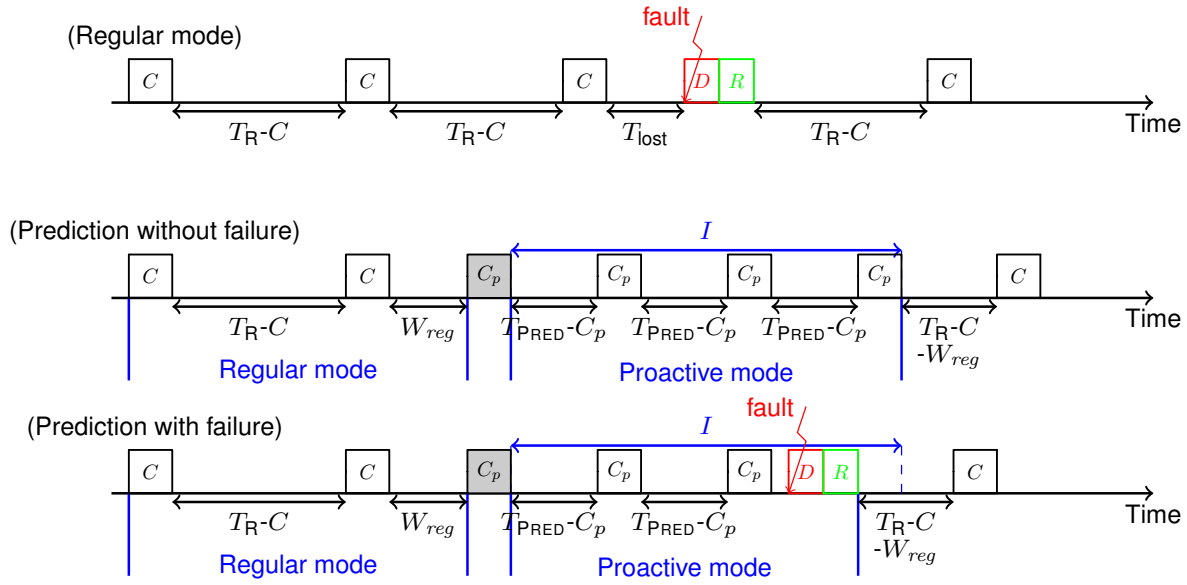


Figure 3.1: Outline of strategy WITHCKPTI.

3. If we decide to trust the predictor, we use various strategies, depending upon the length I of the prediction window.

Before describing the different strategies in situation (3), we point out that the rationale for not always trusting the predictor is to avoid taking useless checkpoints too frequently. Indeed, the precision p of the predictor must be above a given threshold for its usage to be worthwhile. In other words, if we decide to checkpoint just before a predicted event, either we will save time by avoiding a costly re-execution if the event does correspond to an actual fault, or we will lose time by unduly performing an extra checkpoint.

Now, to describe the strategies used when we trust a prediction (situation (3)), we define two *modes* for the scheduling algorithm. The **Regular** mode is used when no fault prediction is available, or when a prediction is available but we decide to ignore it (with probability $1 - q$). In regular mode, we use periodic checkpointing with period T_R. Intuitively, T_R corresponds to the checkpointing period T of Chapter 2. The **Proactive** mode is used when a fault prediction is available and we decide to trust it, a decision taken with probability q. Consider such a trusted prediction made for a prediction window $[t_0, t_0 + I]$. Several strategies can be envisioned:

(1) WITHCKPTI, for *With checkpoints during prediction window*– The first strategy (see Figure 3.1) is intended for long prediction window and assumes that $C_p \leq I$: the algorithm interrupts the current period (of scheduled length T_R), and checkpoints during the interval $[t_0 - C_p, t_0]$, and decides to take several checkpoints during the prediction window. The period T_{PRED} of these checkpoints in proactive mode will presumably be shorter than T_R, to take into account the higher fault probability. In the following, we analytically compute the optimal number of such periods. But we assume that there is at least one period here, hence, that we take at least one checkpoint (in the absence of faults), which implies $C_p \leq I$. We return to regular mode either right after the fault strikes within the time window $[t_0, t_0 + I]$, or at time $t_0 + I$ if no actual fault happens within this window. Then, we resume the work needed to complete the interrupted period of the regular mode. The first strategy is the most complex to describe, and the complete behavior of the corresponding scheduling algorithm is shown in Algorithm 1.

(2) NOCKPTI, for *No checkpoint during prediction window*– The second strategy (see Figure 3.2) is intended for a shorter prediction window: we still acknowledge it, but make the decision not to check-

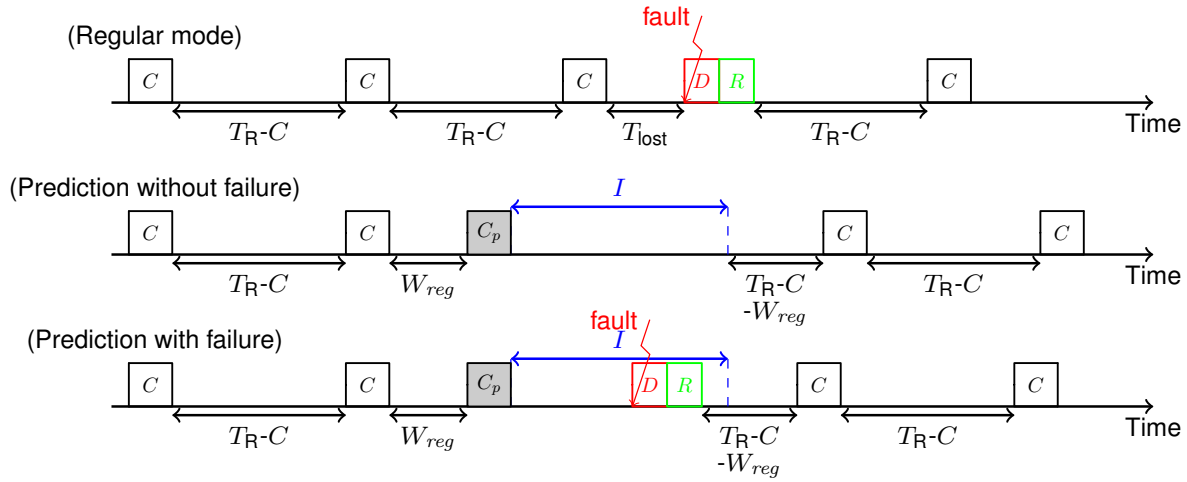


Figure 3.2: Outline of strategy NOCKPTI.

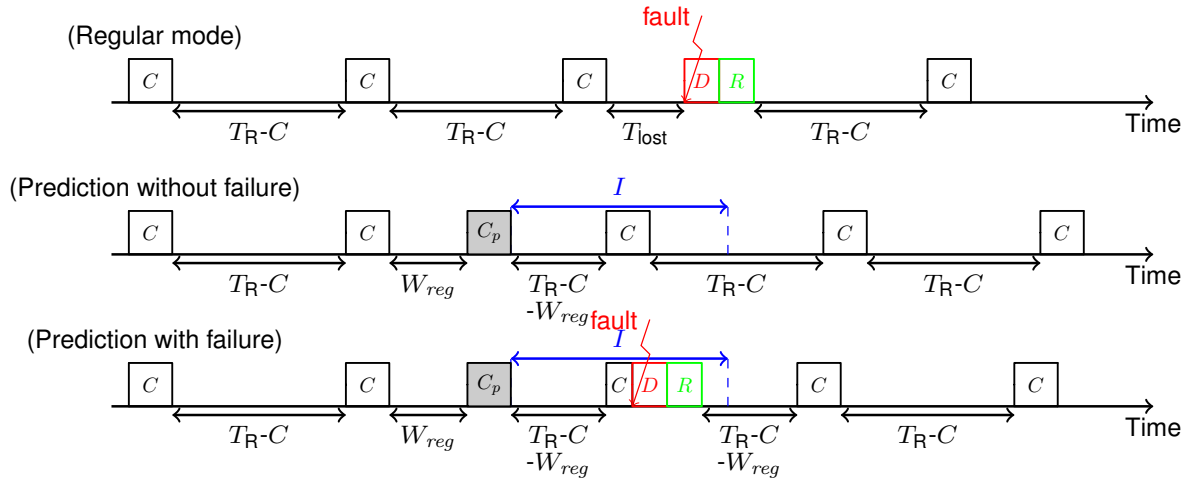


Figure 3.3: Outline of strategy INSTANT.

point during it. The algorithm interrupts the current period (of scheduled length T_R), checkpoints during the interval $[t_0 - C_p, t_0]$, and then returns to regular mode: at time $t_0 + I$, it resumes the work needed to complete the interrupted period of the regular mode.
During the whole length of the time-window, we execute work without checkpointing, at the risk of losing work if a fault indeed strikes. But for a small value of I, it may not be worthwhile to checkpoint during the prediction window (if at all possible, since there is no choice if $I < C_p$).

(3) INSTANT, for *Instantaneous*– The second strategy (see Figure 3.3) is to ignore the time-window and to execute the same algorithm as if the predictor had given an exact date prediction at time t_0. As in the second strategy, the algorithm interrupts the current period (of scheduled length T_R), and checkpoints during the interval $[t_0 - C_p, t_0]$. But here, we return to regular mode at time t_0, where we resume the work needed to complete the interrupted period of the regular mode.

Note that, for all strategies, we insert some additional work for the particular case where there is not enough time to take a checkpoint before entering proactive mode (because a checkpoint for the regular

mode is currently on-going). We account for this work as idle time in the expression of the waste, to ease the analysis. Our expression of the waste is thus an upper bound.

Algorithm 1: WITHCKPTI

if *fault happens* **then**
  After downtime, execute recovery;
  Enter *regular* mode;
if *in* proactive *mode for a time greater than or equal to* I **then**
  Switch to *regular* mode;
if *Prediction made with interval* $[t, t+I]$ ***and*** *prediction taken into account* **then**
  Let t_C be the date of the last checkpoint under *regular* mode to start no later than $t - C_p$;
  if $t_C + C < t - C_p$ **then** (enough time for extra checkpoint)
    Take a checkpoint starting at time $t - C_p$
  else (no time for the extra checkpoint)
    Work in the time interval $[t_C + C, t]$
  $W_{reg} \leftarrow \max(0, t - C_p - (t_C + C))$;
  Switch to *proactive* mode at time t;
while *in* regular *mode and no predictions are made and no faults happen* **do**
  Work for a time T_{R}-W_{reg}-C and then checkpoint;
  $W_{reg} \leftarrow 0$;
while *in* proactive *mode and no faults happen* **do**
  Work for a time T_{PRED}-C_p and then checkpoint;

3.3.1 Strategy WITHCKPTI

In this section, we evaluate the execution time under heuristic WITHCKPTI. To do so, we partition the whole execution into time intervals defined by the presence or absence of events. An interval starts and ends with either the completion of a checkpoint or of a recovery (after a failure). To ease the analysis, we make a simplifying hypothesis: we assume that *at most* one event, failure or prediction, occurs within any interval of length $T_{\mathrm{R}} + I + C_p$. In particular, this implies that a prediction or an unpredicted fault always take place during the regular mode.

We list below the four types of intervals, and evaluate their respective average length, together with the average work completed during each of them (see Table 3.2 for a summary):

1. **Two consecutive regular checkpoints with no intermediate events.** The time elapsed between the completion of the two checkpoints is exactly T_{R}, and the work done is exactly $T_{\mathrm{R}} - C$.
2. **Unpredicted fault.** Recall that, because of the simplifying hypothesis, the fault happens in regular mode. Because instants where the fault strikes and where the last checkpoint was taken are independent, on average the fault strikes at time $T_{\mathrm{R}}/2$. A downtime of length D and a recovery of length R occur before the interval completes. There is no work done.
3. **False prediction.** Recall that it happens in regular mode. There are two cases:
   (a) **Taken into account.** This happens with probability q. The interval lasts $T_{\mathrm{R}} + C_p + I$, since we take a proactive checkpoint and spend the time I in proactive mode. The work done is $(T_{\mathrm{R}} - C) + (I - \frac{I}{T_{\mathrm{PRED}}} C_p)$.

Mode	Number of intervals	Time spent	Work done
(1)	w_1	T_{R}	$T_{\mathrm{R}} - C$
(2)	$w_2 = \frac{\textsc{Time}_{\textsc{Final}}}{\mu_{\mathrm{NP}}}$	$T_{\mathrm{R}}/2 + D + R$	0
(3)	$w_3 = \frac{(1-p)\textsc{Time}_{\textsc{Final}}}{\mu_{\mathrm{P}}}$	$T_{\mathrm{R}} + q(I + C_p)$	$T_{\mathrm{R}} - C + q(I - \frac{I}{T_{\textsc{Pred}}} C_p)$
(4)	$w_4 = \frac{p\textsc{Time}_{\textsc{Final}}}{\mu_{\mathrm{P}}}$	$q(T_{\mathrm{R}} + \mathbb{E}_I^{(f)} + C_p)$ $+(1-q)T_{\mathrm{R}}/2 + D + R$	$q\left(T_{\mathrm{R}} - C + \left(\frac{\mathbb{E}_I^{(f)}}{T_{\textsc{Pred}}} - 1\right)(T_{\textsc{Pred}} - C_p)\right)$

Table 3.2: Summary of the different interval types for WITHCKPTI.

(b) **Not taken into account.** This happens with probability $1 - q$. The interval lasts T_{R} and the work done is $T_{\mathrm{R}} - C$.

Considering both cases with their probabilities, the average time spent is equal to: $q(T_{\mathrm{R}} + C_p + I) + (1-q)T_{\mathrm{R}} = T_{\mathrm{R}} + q(C_p + I)$. The average work done is: $q(T_{\mathrm{R}} - C + I - \frac{I}{T_{\textsc{Pred}}} C_p) + (1 - q)(T_{\mathrm{R}} - Cr) = T_{\mathrm{R}} - C + q(I - \frac{I}{T_{\textsc{Pred}}} C_p)$.

4. **True prediction.** Recall that it happens in regular mode. There are two cases:
   (a) **Taken into account.** Let $\mathbb{E}_I^{(f)}$ be the average time at which a fault occurs within the prediction window (the time at which the fault strikes is certainly correlated to the starting time of the prediction window; $\mathbb{E}_I^{(f)}$ may not be equal to $I/2$). Up to time $\mathbb{E}_I^{(f)}$, we work and checkpoint in proactive mode, with period $T_{\textsc{Pred}}$. In addition, we take a proactive checkpoint right before the start of the prediction window. Then we spend the time $\mathbb{E}_I^{(f)}$ in proactive mode, and we have a downtime and a recovery. Hence, such an interval lasts $T_{\mathrm{R}} + C_p + \mathbb{E}_I^{(f)} + D + R$ on average. The total work done during the interval is $T_{\mathrm{R}} - C + x(T_{\textsc{Pred}} - C_p)$ where x is the expectation of the number of proactive checkpoints successfully taken during the prediction window. Here, $x \approx \frac{\mathbb{E}_I^{(f)}}{T_{\textsc{Pred}}} - 1$.
   (b) **Not taken into account.** On average the fault occurs at time $T_{\mathrm{R}}/2$. The time interval has duration $T_{\mathrm{R}}/2 + D + R$, and there is no work done.

   Overall the time spent is $q(T_{\mathrm{R}} + C_p + \mathbb{E}_I^{(f)} + D + R) + (1-q)(T_{\mathrm{R}}/2 + D + R)$, and the work done is $q(T_{\mathrm{R}} - C + (\frac{\mathbb{E}_I^{(f)}}{T_{\textsc{Pred}}} - 1)(T_{\textsc{Pred}} - C_p)) + (1-q)0$.

We want to estimate the total execution time, $\textsc{Time}_{\textsc{Final}}$. So far, we have evaluated the length, and the work done, for each of the interval types. We now estimate the expectation of the number of intervals of each type. Consider the intervals defined by an event whose mean time between occurrences is ν. On average, during a time T, there will be T/ν such intervals. Due to the simplifying hypothesis, intervals of different types never overlap. Table 3.2 presents the estimation of the number of intervals of each type.

To estimate the time spent within intervals of a given type, we multiply the expectation of the number of intervals of that type by the expectation of the time spent in each of them. Of course, multiplying expectations is correct only if the corresponding random variables are independent. Nevertheless, we hope that this will lead to a good approximation of the expected execution time. We will assess the quality of the approximation through simulations in Section 3.4. We have:

$$\textsc{Time}_{\textsc{Final}} = w_1 \times T_{\mathrm{R}} + w_2 \left(\frac{T_{\mathrm{R}}}{2} + D + R\right) + w_3 \left(T_{\mathrm{R}} + q(I + C_p)\right) + w_4 \left(q(T_{\mathrm{R}} + \mathbb{E}_I^{(f)} + C_p) + (1-q)\frac{T_{\mathrm{R}}}{2} + D + R\right). \quad (3.1)$$

We use the same line of reasoning to compute the overall amount of work done, that must be equal, by definition, to $\text{TIME}_{\text{base}}$, the execution time of the application without any overhead:

$$\begin{aligned}\text{TIME}_{\text{base}} =& w_1(T_{\text{R}} - C) + w_2 \times 0 + w_3 \left(T_{\text{R}} - C + q\left(I - \frac{I}{T_{\text{PRED}}}C_p\right)\right) \\ &+ w_4 \left(q\left(T_{\text{R}} - C + \left(\frac{\mathbb{E}_I^{(f)}}{T_{\text{PRED}}} - 1\right)(T_{\text{PRED}} - C_p)\right)\right).\end{aligned} \tag{3.2}$$

This equation gives the value of w_1 as a function of the other parameters. Looking at Equations (3.1) and (3.2), and at the values of w_2, w_3, and w_4, we remark that $\text{TIME}_{\text{Final}}$ can be rewritten as a function of q as follows: $\text{TIME}_{\text{Final}} = \alpha\text{TIME}_{\text{base}} + \beta\text{TIME}_{\text{Final}} + q\gamma\text{TIME}_{\text{Final}}$, that is $\text{TIME}_{\text{Final}} = \frac{\alpha}{1-\beta-q\gamma}\text{TIME}_{\text{base}}$, where neither α, nor β, nor γ depend on q. The derivative of $\text{TIME}_{\text{Final}}$ with respect to q has constant sign. Hence, in an optimal solution, either $q = 0$ or $q = 1$. This (somewhat unexpected) conclusion is that the predictor should sometimes be always trusted, and sometimes never, but no in-between value for q will do a better job. Thus we can now focus on the two functions $\text{TIME}_{\text{Final}}$, the one when $q = 0$ ($\text{TIME}_{\text{Final}}^{\{0\}}$), and the one when $q = 1$ ($\text{TIME}_{\text{Final}}^{\{1\}}$).

When $q = 0$, from Table 3.2 and Equations (3.1) and (3.2), we derive that

$$\begin{aligned}\text{TIME}_{\text{Final}}^{\{0\}} &= \frac{T_{\text{R}}}{T_{\text{R}} - C}\text{TIME}_{\text{base}} + \frac{\text{TIME}_{\text{Final}}^{\{0\}}}{\mu}\left(\frac{T_{\text{R}}}{2} + D + R\right), \\ \left(1 - \frac{C}{T_{\text{R}}}\right)&\left(1 - \frac{T_{\text{R}}/2 + D + R}{\mu}\right)\text{TIME}_{\text{Final}}^{\{0\}} = \text{TIME}_{\text{base}}.\end{aligned} \tag{3.3}$$

This is exactly the equation from Chapter 2 in the case of exact-date predictions that are never taken into account (a good sanity check!). When $q = 1$, we have:

$$\begin{aligned}\text{TIME}_{\text{Final}}^{\{1\}} =& \text{TIME}_{\text{base}}\frac{T_{\text{R}}}{T_{\text{R}} - C} \\ &- \frac{\text{TIME}_{\text{Final}}^{\{1\}}}{\mu_{\text{P}}}\frac{T_{\text{R}}}{T_{\text{R}} - C}\left((T_{\text{R}} - C) + (1-p)\left(I - \frac{I}{T_{\text{PRED}}}C_p\right) + p\left(\frac{\mathbb{E}_I^{(f)}}{T_{\text{PRED}}} - 1\right)(T_{\text{PRED}} - C_p)\right) \\ &+ \frac{\text{TIME}_{\text{Final}}^{\{1\}}}{\mu_{\text{NP}}}\left(\frac{T_{\text{R}}}{2} + D + R\right) + \frac{(1-p)\text{TIME}_{\text{Final}}^{\{1\}}}{\mu_{\text{P}}}(T_{\text{R}} + I + C_p) \\ &+ \frac{p\text{TIME}_{\text{Final}}^{\{1\}}}{\mu_{\text{P}}}\left(T_{\text{R}} + C_p + \mathbb{E}_I^{(f)} + D + R\right).\end{aligned}$$

After a little rewriting we obtain:

$$\begin{aligned}\frac{\text{TIME}_{\text{base}}}{\text{TIME}_{\text{Final}}^{\{1\}}} =& \frac{r}{p\mu}\left(1 - \frac{C_p}{T_{\text{PRED}}}\right)\left((1-p)I + p\left(\mathbb{E}_I^{(f)} - T_{\text{PRED}}\right)\right) \\ &+ \left(1 - \frac{C}{T_{\text{R}}}\right)\left(1 - \frac{1}{p\mu}\left(p(D+R) + rC_p + (1-r)p\frac{T_{\text{R}}}{2} + r\left((1-p)I + p\mathbb{E}_I^{(f)}\right)\right)\right).\end{aligned}$$

Finally, the waste is equal by definition to $\frac{\text{TIME}_{\text{Final}} - \text{TIME}_{\text{base}}}{\text{TIME}_{\text{Final}}}$. Therefore, we have:

$$\begin{aligned}\text{WASTE} = 1 &- \frac{r}{p\mu}\left(1 - \frac{C_p}{T_{\text{PRED}}}\right)\left((1-p)I + p\left(\mathbb{E}_I^{(f)} - T_{\text{PRED}}\right)\right) \\ &- \left(1 - \frac{C}{T_{\text{R}}}\right)\left(1 - \frac{1}{p\mu}\left(p(D+R) + rC_p + (1-r)p\frac{T_{\text{R}}}{2} + r\left((1-p)I + p\mathbb{E}_I^{(f)}\right)\right)\right).\end{aligned} \tag{3.4}$$

Mode	Number of intervals	Time spent	Work done
(1)	w_1	T_{R}	$T_{\mathrm{R}} - C$
(2)	$w_2 = \frac{\textsc{Time}_{\mathrm{Final}}}{\mu_{\mathrm{NP}}}$	$T_{\mathrm{R}}/2 + D + R$	0
(3)	$w_3 = \frac{(1-p)\textsc{Time}_{\mathrm{Final}}}{\mu_{\mathrm{P}}}$	$T_{\mathrm{R}} + q(I + C_p)$	$T_{\mathrm{R}} - C + qI$
(4)	$w_4 = \frac{p\textsc{Time}_{\mathrm{Final}}}{\mu_{\mathrm{P}}}$	$q(T_{\mathrm{R}} + \mathbb{E}_I^{(f)} + C_p)$ $+(1-q)T_{\mathrm{R}}/2 + D + R$	$q\,(T_{\mathrm{R}} - C)$

Table 3.3: Summary of the different interval types for NoCkptI.

Waste minimization

When $q = 0$, the optimal period can readily be computed from Equation (3.3) and we derive that the optimal period is $\sqrt{2(\mu - (D+R))C}$. This defines a periodic policy we call RFO, for Refined First-Order approximation. We now minimize the waste of the strategy where $q = 1$. In order to compute the optimal value for $T_{\textsc{Pred}}$, we identify the fraction of the waste in Equation (3.4) that depends on $T_{\textsc{Pred}}$. We can rewrite Equation (3.4) as:

$$\textsc{Waste}^{\{1\}} = \alpha + \frac{r}{p\mu}\left(\left((1-p)\,I + p\mathbb{E}_I^{(f)}\right)\frac{C_p}{T_{\textsc{Pred}}} + pT_{\textsc{Pred}}\right)$$

where α does not depend on $T_{\textsc{Pred}}$. The waste is thus minimized when $T_{\textsc{Pred}}$ is equal to $T_{\textsc{Pred}}^{\mathrm{extr}} = \sqrt{\frac{\left((1-p)I + p\mathbb{E}_I^{(f)}\right)C_p}{p}}$. Note that we always have to enforce that $T_{\textsc{Pred}}$ is larger than C_p and does not exceed I. Therefore, the optimal period $T_{\textsc{Pred}}^{\mathrm{opt}}$ is defined as follows: $T_{\textsc{Pred}}^{\mathrm{opt}} = \min\{I,\ \max\{C_p,\ T_{\textsc{Pred}}^{\mathrm{extr}}\}\}$. The rounding only occurs for extreme cases.

In order to compute the optimal value for T_{R}, we identify the fraction of the waste in Equation (3.4) that depends on T_{R}. We can rewrite Equation (3.4) as:

$$\textsc{Waste}^{\{1\}} = \beta + \frac{C}{T_{\mathrm{R}}}\left(1 - \frac{1}{p\mu}\left(p(D+R) + r\left(C_p + (1-p)\,I + p\mathbb{E}_I^{(f)}\right)\right)\right) + \frac{1-r}{\mu}\frac{T_{\mathrm{R}}}{2} \quad (3.5)$$

where β does not depend on T_{R} because $T_{\textsc{Pred}}^{\mathrm{opt}}$ does not depend on T_{R}. Therefore, $\textsc{Waste}^{\{1\}}$ is minimized when T_{R} is equal to

$$T_{\mathrm{R}}^{\mathrm{extr}} = \sqrt{\frac{2C\left(p\mu - \left(p(D+R) + r\left(C_p + \left((1-p)\,I + p\mathbb{E}_I^{(f)}\right)\right)\right)\right)}{p(1-r)}}. \quad (3.6)$$

Recall that we must always enforce that $T_{\mathrm{R}}^{\mathrm{opt}}$ is always greater than C. Also, note that when $r = 0$, we do obtain the same period as without a predictor. Finally, if we assume that, on average, fault strikes at the middle of the prediction window, i.e., $\mathbb{E}_I^{(f)} = \frac{I}{2}$, we obtain simplified values:

$$T_{\textsc{Pred}}^{\mathrm{extr}} = \sqrt{\frac{(2-p)IC_p}{p}} \text{ and } T_{\mathrm{R}}^{\mathrm{extr}} = \sqrt{\frac{2C\left(p\mu - \left(p(D+R) + r\left(C_p + \left(1 - \frac{p}{2}\right)I\right)\right)\right)}{p(1-r)}}.$$

Mode	Number of intervals	Time spent	Work done
(1)	w_1	T_{R}	$T_{\mathrm{R}} - C$
(2)	$w_2 = \frac{\textsc{Time}_{\mathrm{Final}}}{\mu_{\mathrm{NP}}}$	$T_{\mathrm{R}}/2 + D + R$	0
(3)	$w_3 = \frac{(1-p)\textsc{Time}_{\mathrm{Final}}}{\mu_{\mathrm{P}}}$	$T_{\mathrm{R}} + qC_p$	$T_{\mathrm{R}} - C$
(4)	$w_4 = \frac{p\textsc{Time}_{\mathrm{Final}}}{\mu_{\mathrm{P}}}$	$q(T_{\mathrm{R}} + \mathbb{E}_I^{(f)} + C_p)$ $+(1-q)T_{\mathrm{R}}/2 + D + R$	$q\,(T_{\mathrm{R}} - C)$

Table 3.4: Summary of the different interval types for INSTANT.

3.3.2 Strategy NOCKPTI

In this section, we evaluate the execution time under heuristic NOCKPTI. For clarity, we only summarize results and refer to [RR8] for details. The analysis is similar to that for WITHCKPTI. Table 3.3 provides the estimation of the number of intervals of each type. As for WITHCKPTI, one shows that in an optimal solution, either $q = 0$ or $q = 1$. When $q = 0$, we derive that

$$\textsc{Time}_{\mathrm{Final}}^{\{0\}} = \frac{T_{\mathrm{R}}}{T_{\mathrm{R}} - C}\textsc{Time}_{\mathrm{base}} + \frac{\textsc{Time}_{\mathrm{Final}}^{\{0\}}}{\mu}\left(\frac{T_{\mathrm{R}}}{2} + D + R\right),$$

$$\left(1-\frac{C}{T_{\mathrm{R}}}\right)\left(1 - \frac{T_{\mathrm{R}}/2 + D + R}{\mu}\right)\textsc{Time}_{\mathrm{Final}}^{\{0\}} = \textsc{Time}_{\mathrm{base}}. \tag{3.7}$$

This is exactly the equation from Chapter 2 in the case of exact-date predictions that are never taken into account, what we had already retrieved with WITHCKPTI (same sanity check!). When $q = 1$, we derive that:

$$\begin{aligned}\textsc{Waste} = 1 - \frac{r}{p\mu}(1-p)I - \left(1-\frac{C}{T_{\mathrm{R}}}\right) \times \\ \left(1-\frac{1}{p\mu}\left(p(D+R)+rC_p+(1-r)p\frac{T_{\mathrm{R}}}{2}+r\left((1-p)I+p\mathbb{E}_I^{(f)}\right)\right)\right).\end{aligned} \tag{3.8}$$

Waste minimization

The waste is minimized as follows:

- When $q = 0$, the optimal value for T_{R} is the same as the one we computed for WITHCKPTI in the case $q = 0$.
- When $q = 1$, the value of T_{R} that minimizes the waste is $T_{\mathrm{R}}^{\mathrm{extr}}$, the value given by Equation (3.6).

3.3.3 Strategy INSTANT

In this section we evaluate the execution time under heuristic NOCKPTI. For clarity, we only summarize results and refer to [RR8] for details. The analysis is similar to the previous ones. Table 3.4 provides the estimation of the number of intervals of each type. As before, one shows that in an optimal solution, either $q = 0$ or $q = 1$. When $q = 0$, we derive, once again, that

$$\textsc{Time}_{\mathrm{Final}}^{\{0\}} = \frac{T_{\mathrm{R}}}{T_{\mathrm{R}} - C}\textsc{Time}_{\mathrm{base}} + \frac{\textsc{Time}_{\mathrm{Final}}^{\{0\}}}{\mu}\left(\frac{T_{\mathrm{R}}}{2} + D + R\right),$$

$$\left(1-\frac{C}{T_{\mathrm{R}}}\right)\left(1-\frac{T_{\mathrm{R}}/2+D+R}{\mu}\right)\textsc{Time}_{\mathrm{Final}}^{\{0\}}=\textsc{Time}_{\mathrm{base}}. \tag{3.9}$$

This is exactly the equation from Chapter 2 in the case of exact-date predictions that are never taken into account, what we had already with WITHCKPTI and NOCKPTI (yet another good sanity check!). When $q=1$, we obtain

$$\begin{aligned}\textsc{Waste}=1-&\left(1-\frac{C}{T_{\mathrm{R}}}\right)\times\\&\left(1-\frac{1}{p\mu}\left(p(D+R)+rC_p+(1-r)p\frac{T_{\mathrm{R}}}{2}+pr\mathbb{E}_I^{(f)}\right)\right).\end{aligned} \tag{3.10}$$

Waste minimization

The waste is minimized as follows:

- When $q=0$, the optimal value for T_{R} is the same as for WITHCKPTI and for NOCKPTI in the case $q=0$.
- When $q=1$, the optimal value for T_{R} is

$$T_{\mathrm{R}}^{\mathrm{extr}}=\sqrt{\frac{2C\left(p\mu-\left(p(D+R)+rC_p+pr\mathbb{E}_I^{(f)}\right)\right)}{p(1-r)}}.$$

  Again, recall that we must always enforce that $T_{\mathrm{R}}^{\mathrm{opt}}$ is always greater than C. Finally, if we assume that, on average, fault strikes at the middle of the prediction window, i.e., $\mathbb{E}_I^{(f)}=\frac{I}{2}$, we have:

$$T_{\mathrm{R}}^{\mathrm{extr}}=\sqrt{\frac{2C\left(p\mu-\left(p(D+R)+rC_p+pr\frac{I}{2}\right)\right)}{p(1-r)}}.$$

3.4 Simulation results

An experimental validation of the models at targeted scale would require running a large application several times, for each checkpointing strategy, for each fault predictor, and for each platform size. This would require a prohibitive amount of computational hours. Furthermore, some of the targeted platform sizes currently exist only as reasonable projections. Therefore, we resort to simulations. We present the simulation framework in Section 3.4.1. Then we report results using the characteristics of two fault predictors (Section 3.4.2). Additional figures and data results are available in the research report [RR8].

3.4.1 Simulation framework

In order to validate the model, we have instantiated it with several scenarios. The simulations use parameters that are representative of current and forthcoming large-scale platforms [25, 43]. We take $C=R=600$ seconds, and $D=60$ seconds. We consider three scenarios where proactive checkpoints are (i) exactly as expensive as periodic checkpoints ($C_p=C$); (ii) ten times cheaper ($C_p=0.1C$); and (iii) two times more expensive ($C_p=2C$). The individual (processor) MTBF is $\mu_{\mathrm{ind}}=125$ years, and the total number of processors N varies from $N=2^{16}=16,384$ to $N=2^{19}=524,288$, so that the platform MTBF μ varies from $\mu=4,010$ min (about 2.8 days) down to $\mu=125$ min (about 2 hours). For instance the Jaguar platform, with $N=45,208$ processors, is reported to have experienced

about one fault per day [128], which leads to $\mu_{\text{ind}} = \frac{45,208}{365} \approx 125$ years. The application size is set to $\text{TIME}_{\text{base}} = 10,000$ years/N.

We use Maple to analytically compute and plot the optimal value of the waste for the three prediction-aware policies, INSTANT, NOCKPTI, and WITHCKPTI, for the prediction-ignoring policy RFO (corresponding to the case $q = 0$), and for the reference heuristic DALY (Daly's [35] periodic policy). In order to check the accuracy of our model, we have compared the analytical results with results obtained with a discrete-event simulator. The simulation engine generates a random trace of faults, parameterized either by an Exponential fault distribution or by Weibull distribution laws with shape parameter 0.5 or 0.7. Note that Exponential faults are widely used for theoretical studies, while Weibull faults are representative of the behavior of real-world platforms [58, 110, 59]. In both cases, the distribution is scaled so that its expectation corresponds to the platform MTBF μ. With probability r, we decide if a fault is predicted or not. The simulation engine also generates a random trace of false predictions, whose distribution is identical to that of the first trace (results are similar when false predictions follow a uniform distribution [RR8]). This second distribution is scaled so that its expectation is equal to $\frac{\mu_P}{1-p} = \frac{p\mu}{r(1-p)}$, the inter-arrival time of false predictions. Finally, both traces are merged to produce the final trace including all events (true predictions, false predictions, and non predicted faults). The source code of the fault-simulator and the raw simulation results are freely available at `http://graal.ens-lyon.fr/~fvivien/DATA/predictionwindow/`. Each reported value is the average over 100 randomly generated instances.

In the simulations, we compare the five checkpointing strategies listed above. To assess the quality of each strategy, we compare it with its BESTPERIOD counterpart, defined as the same strategy but using the best possible period T_{R}. This latter period is computed via a brute-force numerical search for the optimal period. Altogether, there are four BESTPERIOD heuristics, one for each of the three variants with prediction, and one for the case where we ignore predictions, which corresponds to both DALY and RFO. Altogether we have a rich set of nine heuristics, which enables us to comprehensively assess the actual quality of the proposed strategies. Note that for computer algebra plots, obviously we do not need BESTPERIOD heuristics, since each period is already chosen optimally from the equations.

We experiment with two predictors from the literature: one accurate predictor with high recall and precision [125], namely with $p = 0.82$ and $r = 0.85$, and another predictor with more limited recall and precision [131], namely with $p = 0.4$ and $r = 0.7$. In both cases, we use five different prediction windows, of size $I = 300$, 600, 900, 1200, and 3000 seconds. Figure 3.4 shows the average waste degradation of the nine heuristics for both predictors, as a function of the number of processors N. We draw the plots as a function of the number of processors N rather than of the platform MTBF $\mu = \mu_{ind}/N$, because it is more natural to see the waste increase with larger platforms; however, this work is agnostic of the granularity of the processors and intrinsically focuses on the impact of the MTBF on the waste.

3.4.2 Analysis of the results

We start with a preliminary remark: when the graphs for INSTANT and WITHCKPTI cannot be seen in the figures, this is because their performance is identical to that of NOCKPTI, and their respective graphs are superposed.

We first compare the analytical results, plotted by the Maple curves, to the simulations results. As shown in Figure 3.4, there is a good correspondence between the analytical curves and the simulations, especially those using an Exponential distribution of failures. However, the larger the platform (or the smaller the MTBF), the less realistic our assumption that no two events happen during an interval of length $T_{\text{R}} + I + C_p$, and the analytical models become less accurate for prediction-aware heuristics.

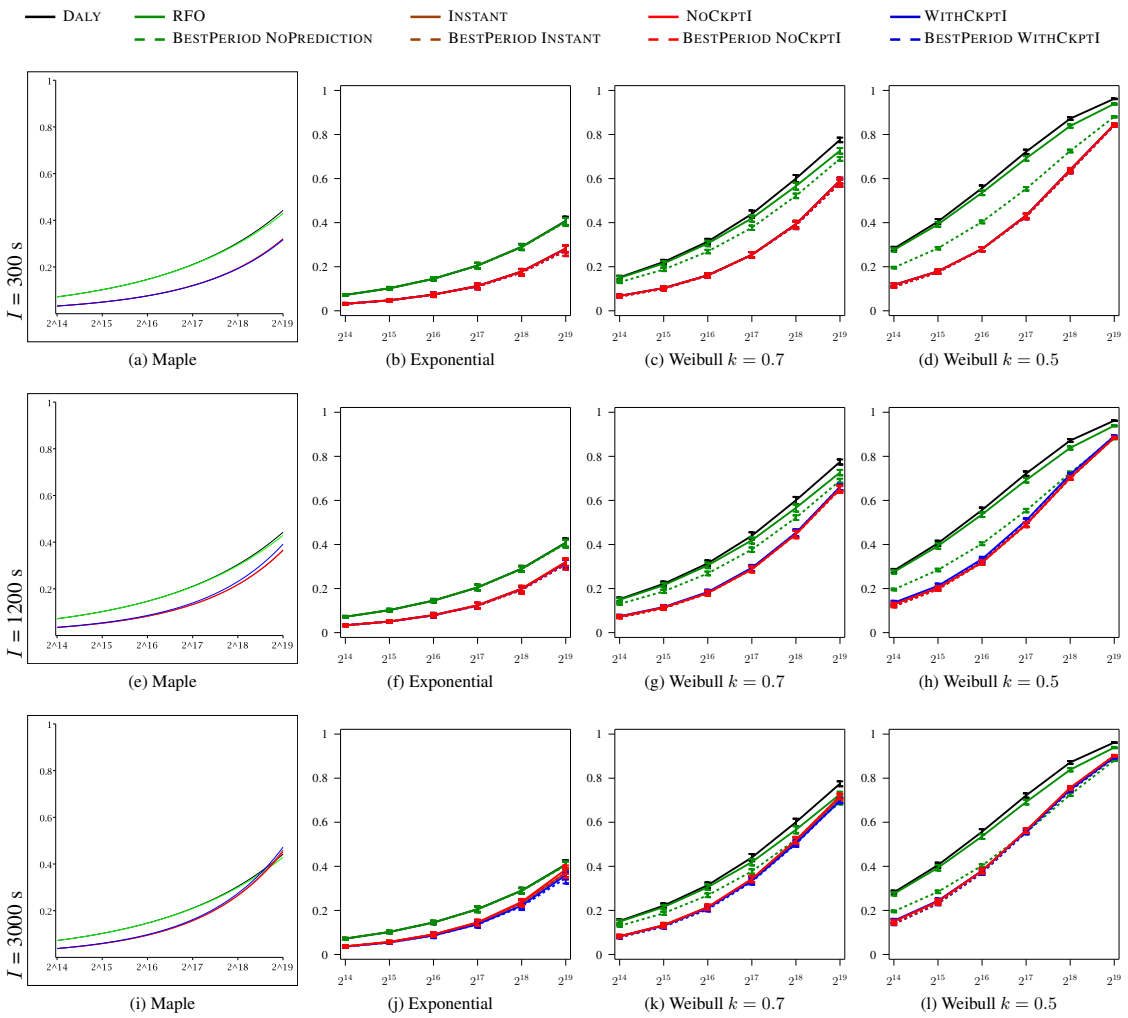
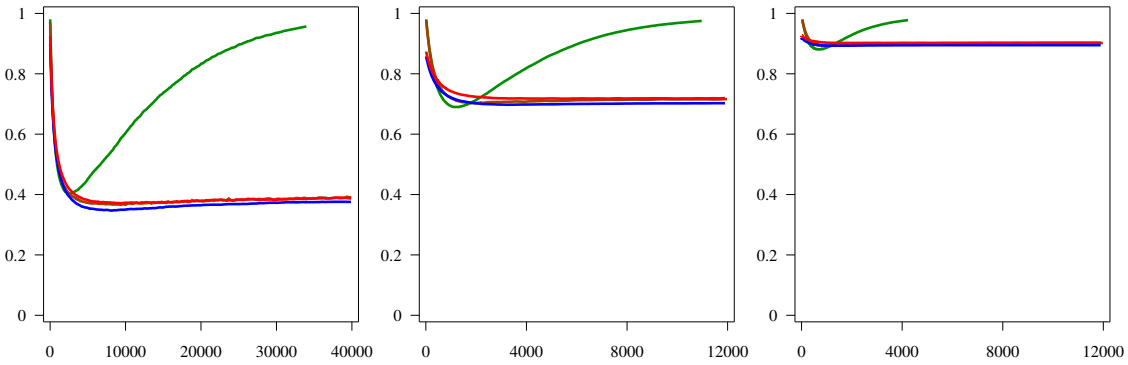
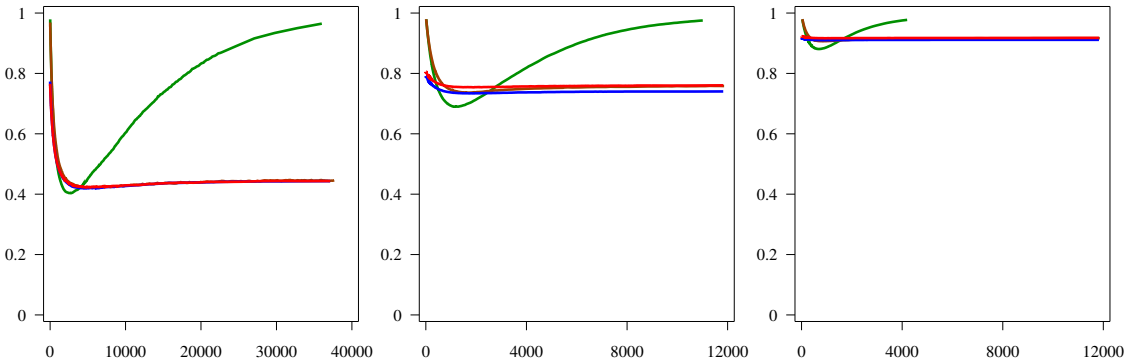


Figure 3.4: Waste as a function of number N of processors, when $p = 0.82$, $r = 0.85$, $C_p = C$.



(a) Exponential (b) Weibull $k = 0.7$ (c) Weibull $k = 0.5$

Figure 3.5: Waste as function of the period T_{R}, with $p = 0.82$, $r = 0.85$, $C_p = C$, $I = 3000s$, and a platform of 2^{19} processors.



(a) Exponential (b) Weibull $k = 0.7$ (c) Weibull $k = 0.5$

Figure 3.6: Waste as function of the period T_{R}, with $p = 0.4$, $r = 0.7$, $C_p = C$, $I = 3000s$, and a platform of 2^{19} processors.

Therefore, the analytical results are overly pessimistic in the most failure-prone platforms. Also, recall that an exponential law is a Weibull law of shape parameter 1. Therefore, the further the distribution of failures is from an exponential law, the larger the difference between analytical results and simulated ones. However, in all cases, the analytical results are able to predict the general *trends*.

A second assessment of the quality of our analysis comes from the BESTPERIOD variants of our heuristics. When predictions are not taken into account, DALY, and to a lesser extent RFO, are not close to the optimal period given by BESTPERIOD (a similar observation was made by Bougeret et al [17]). This gap increases when the distribution is further apart from an Exponential distribution. However, prediction-aware heuristics are very close to BESTPERIOD in almost all configurations. The only exception is with heuristics INSTANT when $C_p = 2C$, the total number of processors N is equal to either 2^{18} and 2^{19}, and I is large. However, when $I = 3000$s and $N = 2^{19}$, the platform MTBF is approximately equal to $6C_p$ which renders our hypothesis and analysis invalid. The difference in this case between INSTANT and its BESTPERIOD should therefore not come as a surprise.

To better understand why close-to-optimal periods are obtained by prediction-aware heuristics (while this is not the case without predictions), we plot the waste as a function of the period T_{R} for RFO and the prediction-aware heuristics (Figure 3.6). On these figures one can see that, whatever the configuration, periodic checkpointing policies (ignoring predictions) have well-defined global optimum. (One should

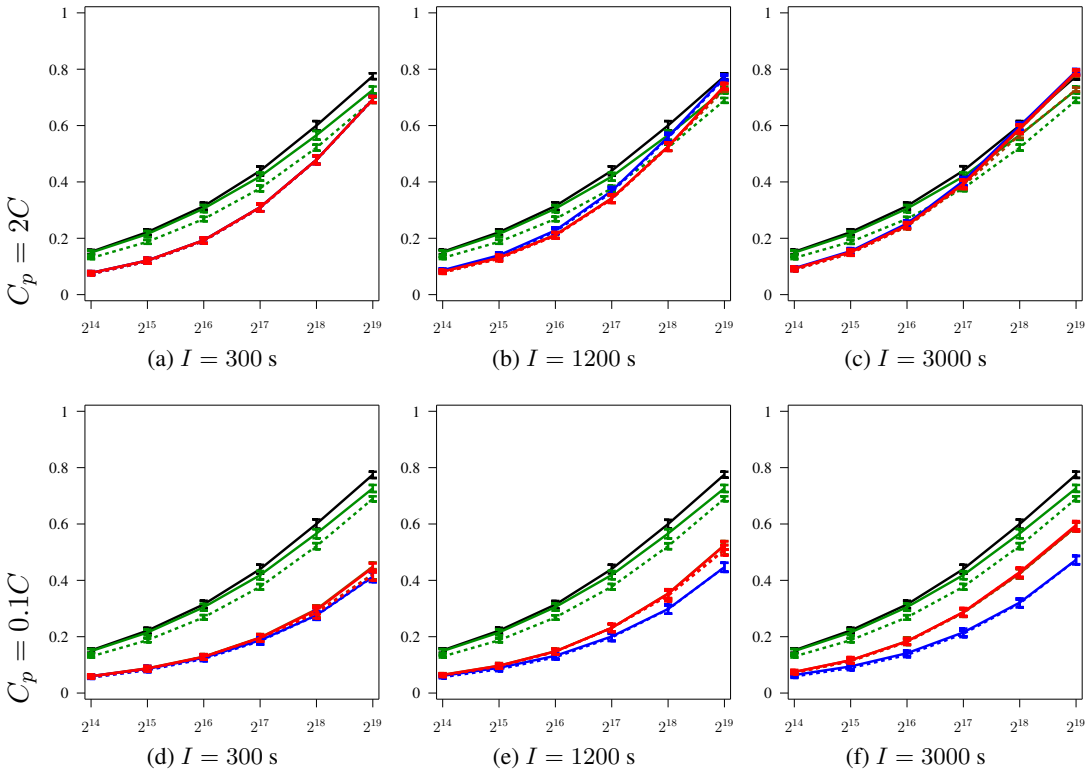


Figure 3.7: Waste with $p = 0.82$, $r = 0.85$, and Weibull law of parameter 0.7.

nevertheless remark that the performance is almost constant in the neighborhood of the optimal period, which explains why policies using different periods can obtain in practice similar performance, as in Bouguerra et al. [18].) For prediction-aware heuristics, however, the behavior is quite different and two scenarios are possible. In the first one, once the optimum is reached, the waste very slowly increases to reach an asymptotic value which is close to the optimum waste (e.g., when the platform MTBF is large and failures follow an exponential distribution). Therefore, any period chosen close to the optimal one, or greater than it, will deliver good quality performance. In the second scenario, the waste decreases until the period becomes larger than the application size, and the waste stays constant. In other words, in these configurations, periodic checkpointing is unnecessary, only proactive actions matter! This striking result can be explained as follows: a significant fraction of the failures are predicted, and thus taken care of, by proactive checkpoints. The impact of unpredicted failures is mitigated by the proactive measures taken for false predictions. To further mitigate the impact of unpredicted faults, the period T_{R} should be significantly shorter than the mean-time between proactive checkpoints, which would induce a lot of waste due to unnecessary checkpoints if the mean-time between unpredicted faults is large with respect to the mean-time between predictions. This greatly restrict the scenarios for which the periodic checkpointing can lead to a significant decrease of the waste.

Figure 3.7 and 3.8 presents a comparison of the checkpointing strategies for different values of C_p and I. When the prediction window I is shorter than the duration C_p of a proactive checkpoint (i.e., when $I = 300$ s and $C_p \geq C = 600$ s), there is no difference between NoCkptI and WithCkptI. When I is small but greater than C_p (say, when I is around $2C_p$), WithCkptI spends most of the prediction window taking a proactive checkpoint and NoCkptI is more efficient. When I becomes "large" with respect to C_p, WithCkptI can become more efficient than NoCkptI, but becomes significantly more efficient only if the proactive checkpoints are significantly shorter than regular (see also Table 3.5).

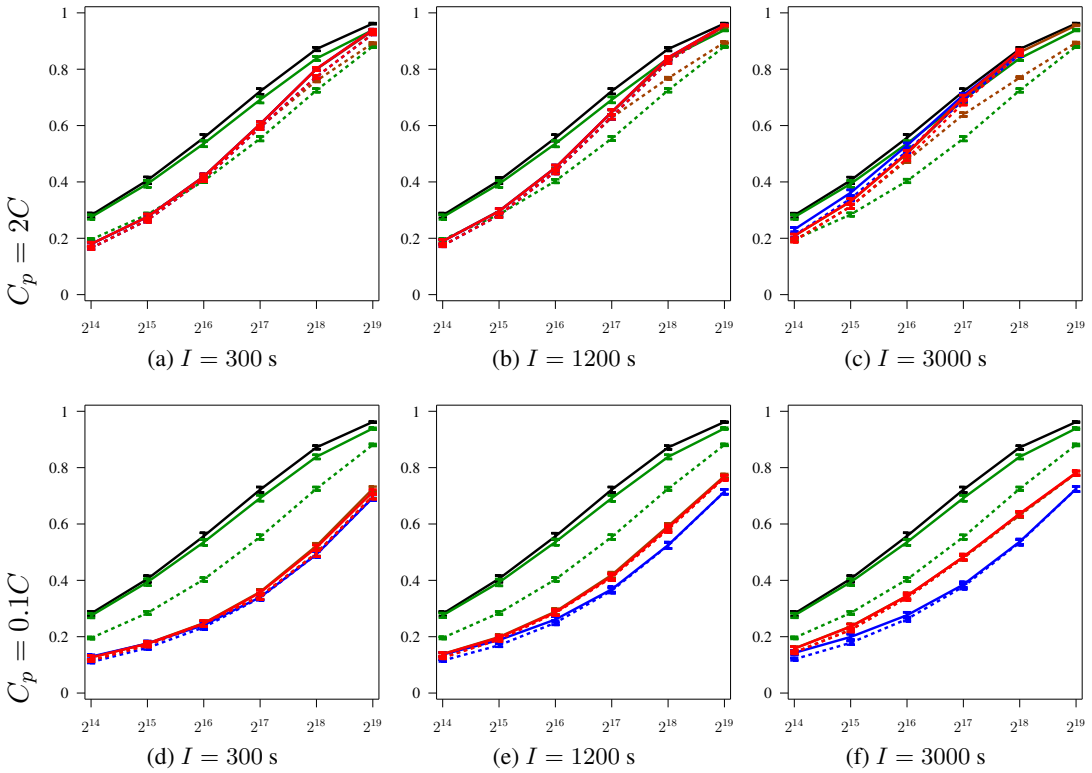


Figure 3.8: Waste with $p = 0.4$, $r = 0.7$, and Weibull law of parameter 0.5.

INSTANT can hardly be seen in the graphs as its performance is most of the time equivalent to that of NOCKPTI.

As expected, the smaller the prediction window, the more efficient the prediction-aware heuristics. Also, the smaller the number of processors (or the larger the platform MTBF), the larger the impact of the size of the prediction window. A surprising result is that taking prediction into account is not always beneficial! The analytical results predict that prediction-aware heuristics would achieve worse performance than periodic policies in our settings, as soon as the platform includes 2^{18} processors. In simulations, results are not so extreme. For the largest platforms considered, using predictions has almost no impact on performance. But when the prediction window is very large, taking predictions into account can indeed be detrimental. These observations can be explained as follows. When the platform includes 2^{19} processors, the platform MTBF is equal to 7500s. Therefore, any interval of duration 3000s has a 40% chance to include a failure: a prediction window of 3000s is not very informative, unless the precision and recall of the predictor are almost equal to 1 (which is never the case in practice). Because the predictor brings almost no knowledge, trusting it may be detrimental. When comparing the performance of, say, NOCKPTI for the two predictors, one can see that when failures follow a Weibull distribution with shape parameter $k = 0.7$, $I = 600$s, and $N = 2^{18}$, NOCKPTI achieves better performance than RFO when $r = 0.85$ and $p = 0.82$, but worse when $p = 0.4$ and $r = 0.7$. The latter predictor generates more false predictions —each one inducing an unnecessary proactive checkpoint— and misses more actual failures —each one destroying some work. The drawbacks of trusting the predictor outweigh the advantages. If failures are few and apart, almost any predictor will be beneficial. When the platform MTBF is small with respect to the cost of proactive checkpoints, only almost perfect predictors will be worth using. For each set of predictor characteristics, there is a threshold for the platform MTBF under which predictions will be useless or detrimental, but above

which predictions will be beneficial.

In order to compare the impact of the heuristics ignoring predictions to those using them, we report job execution times in Table 3.5 when failures follow a Weibull law of parameter 0.7. For each setting, the best performance is presented in bold if it is achieved by a prediction-aware heuristics. For the strategies with prediction, we compute the gain (expressed in percentage) over DALY, the reference strategy without prediction. We first remark that RFO achieves lower makespans than DALY with gains ranging from 1% with 2^{16} processors to 18% with 2^{19} processors. Overall, the gain due to the predictions decreases when the size of the prediction window increases, and increases with the platform size. This gain is obviously closely related to the characteristics of the predictor. When $I = 300$s, the three prediction-aware strategies are identical. When I increases, NOCKPTI achieves slightly better results than INSTANT. For low values of I, WITHCKPTI is the worst prediction-aware heuristics. But when I becomes large and if the predictor is efficient, then WITHCKPTI becomes the heuristics of choice ($I = 3000$s, $p = 0.82$, and $r = 0.85$). The reductions in the application executions times due to the predictor can be very significant. With $p = 0.85$ and $r = 0.82$ and $I = 3000$s, we save 25% of the total time with $N = 2^{19}$, and 13% with $N = 2^{16}$ using strategy WITHCKPTI. With $I = 300$s, we save up to 45% with $N = 2^{19}$, and 18% with $N = 2^{16}$ using any strategy (though NOCKPTI is slightly better than INSTANT). Then, with $p = 0.4$ and $r = 0.7$, we still save 33% of the execution time when $I = 300$s and $N = 2^{19}$, and 14% with $N = 2^{16}$. The gain gets smaller with $I = 3000$s and $N = 2^{16}$ but remains non negligible since we can save 8%. When $I = 3000$s and $N = 2^{19}$, however, the best solution is to ignore predictions and simply use RFO (we fall-back to the case $q = 0$). If we now consider a Weibull law with shape parameter 0.5 instead of 0.7 (see Table 3.6), keeping all other parameters identical ($I = 3000$s, $N = 2^{19}$, $p = 0.4$ and $r = 0.7$), then the heuristics of choice is WITHCKPTI and the gain with respect to DALY is 57.9%.

	$I = 300$ s		$I = 1200$ s		$I = 3000$ s	
	2^{16} procs	2^{19} procs	2^{16} procs	2^{19} procs	2^{16} procs	2^{19} procs
DALY	81.3	31.0	81.3	31.0	81.3	31.0
RFO	80.2 (1%)	25.5 (18%)	80.2 (1%)	25.5 (18%)	80.2 (1%)	25.5 (18%)
$p = 0.82, r = 0.85$						
INSTANT	66.5 (18%)	**17.0** (45%)	68.0 (16%)	20.3 (34%)	70.9 (13%)	24.1 (22%)
NOCKPTI	**66.4** (18%)	**17.0** (45%)	**67.9** (16%)	**20.2** (35%)	71.0 (13%)	24.7 (20%)
WITHCKPTI	**66.4** (18%)	**17.0** (45%)	68.3 (16%)	20.6 (33%)	**70.6** (13%)	**23.1** (25%)
$p = 0.4, r = 0.7$						
INSTANT	70.3 (13%)	20.9 (33%)	72.0 (11%)	24.6 (21%)	**75.0** (8%)	27.7 (11%)
NOCKPTI	**70.2** (14%)	**20.6** (33%)	**71.8** (12%)	**24.2** (22%)	**75.0** (8%)	28.7 (7%)
WITHCKPTI	**70.2** (14%)	**20.6** (33%)	73.6 (9%)	25.5 (18%)	75.1 (8%)	26.6 (14%)

Table 3.5: Job execution times (in days) under the different checkpointing policies for different prediction window size I (in seconds), when failures follow a Weibull distribution of shape parameter 0.7. Gains are reported with respect to DALY.

3.5 Conclusion

In this chapter, we have studied the impact of prediction windows on checkpointing strategies. We have designed several heuristics that decide whether to trust predictions or not, when it is worth taking preventive checkpoints, and at which rate. We have been able to derive a comprehensive set of results and conclusions:

	$I = 300$ s		$I = 1200$ s		$I = 3000$ s	
	2^{16} procs	2^{19} procs	2^{16} procs	2^{19} procs	2^{16} procs	2^{19} procs
DALY	125.7	185.0	125.7	185.0	125.7	185.0
RFO	120.1 (4%)	114.8 (38%)	120.1 (4%)	114.8 (38%)	120.1 (4%)	114.8 (38%)
$p = 0.82$, $r = 0.85$						
INSTANT	**77.4** (38%)	45.2 (76%)	82.0 (35%)	60.8 (67%)	**89.7** (29%)	70.6 (62%)
NOCKPTI	**77.4** (38%)	**44.9** (76%)	**81.8** (35%)	**60.7** (67%)	90.0 (28%)	71.5 (61%)
WITHCKPTI	**77.4** (38%)	**44.9** (76%)	83.6 (33%)	64.4 (65%)	89.8 (29%)	**66.2** (64%)
$p = 0.4$, $r = 0.7$						
INSTANT	84.5 (33%)	59.6 (68%)	89.4 (29%)	76.6 (58%)	**97.7** (22%)	81.9 (56%)
NOCKPTI	**84.4** (33%)	**58.3** (68%)	**89.1** (29%)	76.8 (58%)	97.9 (22%)	83.7 (55%)
WITHCKPTI	**84.4** (33%)	**58.3** (68%)	93.8 (25%)	**75.4** (59%)	97.8 (22%)	**77.7** (58%)

Table 3.6: Job execution times (in days) under the different checkpointing policies for different prediction window size I (in seconds), when failures follow a Weibull distribution of shape parameter 0.5. Gains are reported with respect to DALY.

- We have introduced an analytical model to capture the waste incurred by each strategy, and provided a closed-form formula for each optimization problem, giving the optimal solution. Contrarily to the cases without prediction, or with exact-date predictions, the computation of the waste requires a sophisticated analysis of the various events, including the time spent in the regular and proactive modes.

- The simulations fully validate the model, and the brute-force computation of the optimal period guarantees that our prediction-aware strategies are always very close to the optimal. This holds true both for Exponential and Weibull failure distributions.

- The model is quite accurate and its validity goes beyond the conservative assumption that requires a single event per time interval; even more surprising, the accuracy of the model for prediction-aware strategies is much better than for the case without predictions, where DALY can be far from the optimal period in the case of Weibull failure distributions [17].

- Both the analytical computations and the simulations enable us to characterize when prediction is useful, and which strategy performs better, given the key parameters of the system: recall r, precision p, size of the prediction window I, size of proactive checkpoints C_p versus regular checkpoints C, and platform MTBF μ.

Altogether, the analytical model and the comprehensive results provided in this work enable to fully assess the impact of fault prediction with time-windows on (optimal) checkpointing strategies. Future work will be devoted to refine the assessment of the usefulness of prediction with trace-based failures and prediction logs from current large-scale supercomputers.

Chapter 4

On the combination of silent error detection and checkpointing

4.1 Introduction

Our work in this chapter is motivated by a recent paper by Lu et al. [82], who introduce a *multiple checkpointing model* to compute the optimal checkpointing period with error detection latency. More precisely, Lu et al. [82] deal with the following problem: given errors whose inter arrival times X_e follow an exponential probability distribution of parameter λ_e, and given error detection times X_d that follow an exponential probability distribution of parameter λ_d, what is the optimal checkpointing period T_{opt} in order to minimize the total execution time? The problem is illustrated on Figure 4.1: the error is detected after some (random) time X_d, and one has to rollback up to the last checkpoint that precedes the occurrence of the error. Let k be the number of checkpoints that can be simultaneously kept in memory. Lu et al. [82] derive a formula for the optimal checkpointing period T_{opt} in the (simplified) case where k is unbounded ($k = \infty$), and they propose some numerical simulations to explore the case where k is a fixed constant.

Main contributions. The first major contribution of this chapter is to correct the formula given by Lu et al. [82] when k is unbounded, and to provide an analytical approach when k is a fixed constant. The latter approach is a first-order approximation but applies to any probability distribution of errors.

While it is very natural and interesting to consider the latency of error detection, the model by Lu et al. [82] suffers from an important limitation: it is not clear how one can determine when the error has indeed occurred, and hence how one can identify the last valid checkpoint, unless some verification system is enforced. Another major contribution of this chapter is to introduce a model coupling verification and checkpointing, and to analytically determine the best balance between checkpoints and verifications so as to optimize platform throughput.

The rest of the chapter is organized as follows. First, we revisit the multiple checkpointing model of [82] in Section 4.2; we tackle both the case where all checkpoints are kept, and the case with at most k checkpoints. In Section 4.3, we define and analyze a model coupling checkpoints and verifications. Then, we evaluate the various models in Section 4.4, by instantiating the models with realistic parameters derived from future Exascale platforms. Finally, we conclude and discuss future research directions in Section 4.5.

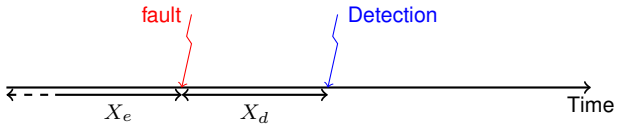


Figure 4.1: Suppose an error (that arrived X_e units of time after the previous error) is detected after a time X_d. Where should we rollback to find a correct checkpoint?

4.2 Revisiting the multiple checkpointing model

In this section, we revisit the approach of Lu et al. [82]. We show that their analysis with unbounded memory is incorrect and provide the exact solution (Section 4.2.1). We also extend their approach to deal with the case where a given (constant) number of checkpoints can be simultaneously kept in memory (Section 4.2.2).

4.2.1 Unlimited checkpoint storage

Let C be the time needed for a checkpoint, R the time for recovery, and D the downtime. Although R and C are a function of the size of the memory footprint of the process, D is a constant that represents the unavoidable costs to rejuvenate a process after an error (e.g., stopping the failed process and restoring a new one that will load the checkpoint image). We assume that errors can take place during checkpoint and recovery but not during downtime (otherwise, the downtime could be considered part of the recovery).

Let $\mu_e = \frac{1}{\lambda_e}$ be the mean time between errors. With no error detection latency and no downtime, well-known formulas for the optimal period (useful work plus checkpointing time that minimizes the execution time) are $T_{\text{opt}} \approx \sqrt{2C\mu_e} + C$ (as given by Young [124]) and $T_{\text{opt}} \approx \sqrt{2C(\mu_e + R)} + C$ (as given by Daly [35]). These formulas are first-order approximations and are valid only if $C, R \ll \mu_e$ (in which case they collapse).

With error detection latency, things are more complicated, even with the assumption that one can track the source of the error (and hence identify the last valid checkpoint). Indeed, the amount of rollback will depend upon the sum $X_e + X_d$. For exponential distributions of X_e and X_d, Lu, Zheng and Chien [82] derive that $T_{\text{opt}} \approx \sqrt{2C(\mu_e + \mu_d)} + C$, where $\mu_d = \frac{1}{\lambda_d}$ is the mean of error detection times. However, although this result may seem intuitive, it is wrong, and we prove that the correct answer is $T_{\text{opt}} \approx \sqrt{2C\mu_e} + C$, even when accounting for the downtime: this first-order approximation is the same as Young's formula. We give an intuitive explanation after the proofs provided in Section 4.2.1. Then in Section 4.2.1, we extend this result to arbitrary laws, but under the additional constraint that $\mu_d + D + R \ll \mu_e$.

Exponential distributions

In this section, we assume that X_e and X_d follow exponential distributions of mean μ_e and μ_d respectively.

Proposition 4.1. *The expected time needed to successfully execute a work of size w followed by its checkpoint is*

$$\mathbb{E}(T(w)) = e^{\lambda_e R}\left(D + \mu_e + \mu_d\right)\left(e^{\lambda_e (w+C)} - 1\right).$$

Proof. Let $T(w)$ be the time needed for successfully executing a work of duration w. There are two cases: (i) if there is no error during execution and checkpointing, then the time needed is exactly $w+C$; (ii) if there is an error before successfully completing the work and its checkpoint, then some additional delays are incurred. These delays come from three sources: the time spent computing by the processors before the error occurs, the time spent before the error is detected, and the time spent for downtime and recovery. Regardless, once a successful recovery has been completed, there still remain w units of work to execute. Thus, we can write the following recursion:

$$\mathbb{E}(T(w)) = e^{-\lambda_e(w+C)}(w+C) + (1-e^{-\lambda_e(w+C)})\left(\mathbb{E}(T_{lost}) + \mathbb{E}(X_d) + \mathbb{E}(T_{rec}) + \mathbb{E}(T(w))\right). \quad (4.1)$$

Here, T_{lost} denotes the amount of time spent by the processors before the first error, knowing that this error occurs within the next $w+C$ units of time. In other terms, it is the time that is wasted because computation and checkpoint were not both completed before the error occurred. The random variable X_d represents the time needed for error detection, and its expectation is $\mathbb{E}(X_d) = \mu_d = \frac{1}{\lambda_d}$. The last variable T_{rec} represents the amount of time needed by the system to perform a recovery. Equation (4.1) simplifies to:

$$\mathbb{E}(T(w)) = w + C + (e^{\lambda_e(w+C)} - 1)(\mathbb{E}(T_{lost}) + \mu_d + \mathbb{E}(T_{rec})). \quad (4.2)$$

We have

$$\begin{aligned}\mathbb{E}(T_{lost}) &= \int_0^\infty x\mathbb{P}(X = x | X < w+C)dx \\ &= \frac{1}{\mathbb{P}(X < w+C)} \int_0^{w+C} x\lambda_e e^{-\lambda_e x} dx,\end{aligned}$$

and $\mathbb{P}(X < w+C) = 1 - e^{-\lambda_e(w+C)}$.
Integrating by parts, we derive that

$$\mathbb{E}(T_{lost}) = \frac{1}{\lambda_e} - \frac{w+C}{e^{\lambda_e(w+C)} - 1}. \quad (4.3)$$

Next, to compute $\mathbb{E}(T_{rec})$, we have a recursive equation quite similar to Equation (4.1) (remember that we assumed that no error can take place during the downtime):

$$\mathbb{E}(T_{rec}) = e^{-\lambda_e R}(D+R) + (1-e^{-\lambda_e R})(\mathbb{E}(R_{lost}) + \mathbb{E}(X_d) + D + \mathbb{E}(T_{rec})).$$

Here, $\mathbb{E}(R_{lost})$ is the expected amount of time lost to executing the recovery before an error happens, knowing that this error occurs within the next R units of time. Replacing $w+C$ by R in Equation (4.3), we obtain

$$\mathbb{E}(R_{lost}) = \frac{1}{\lambda_e} - \frac{R}{e^{\lambda_e R} - 1}.$$

The expression for $\mathbb{E}(T_{rec})$ simplifies to

$$\mathbb{E}(T_{rec}) = De^{\lambda_e R} + (e^{\lambda_e R} - 1)(\mu_e + \mu_d). \quad (4.4)$$

Plugging the values of $\mathbb{E}(T_{lost})$ and $\mathbb{E}(T_{rec})$ into Equation (4.2) leads to the desired value. ■

Proposition 4.2. *The optimal strategy to execute a work of size W is to divide it into n equal-size chunks, each followed by a checkpoint, where n is equal either to $\max(1, \lfloor n^* \rfloor)$ or to $\lceil n^* \rceil$. The value of n^* is uniquely derived from $y = \frac{\lambda_e W}{n^*} - 1$, where $\mathbb{L}(y) = -e^{-\lambda_e C - 1}$ ($\mathbb{L}$, the Lambert function, defined as $\mathbb{L}(x)e^{\mathbb{L}(x)} = x$). The optimal strategy does not depend on the value of μ_d.*

Proof. Using n chunks of size w_i (with $\sum_{i=1}^{n} w_i = W$), by linearity of the expectation, we have $\mathbb{E}(T(W)) = K \sum_{i=1}^{n} (e^{\lambda_e(w_i+C)} - 1)$ where $K = e^{\lambda_e R}(D + \mu_e + \mu_d)$ is a constant. By convexity, the sum is minimum when all the w_is are equal (to $\frac{W}{n}$). Now, $\mathbb{E}(T(W))$ is a convex function of n, hence it admits a unique minimum n^* such that the derivative is zero:

$$e^{\lambda_e(\frac{W}{n^*}+C)}\left(1 - \frac{\lambda_e W}{n^*}\right) = 1. \tag{4.5}$$

Let $y = \frac{\lambda_e W}{n^*} - 1$, we have $ye^y = -e^{-\lambda_e C - 1}$, hence $\mathbb{L}(y) = -e^{-\lambda_e C - 1}$. Then, since we need an integer number of chunks, the optimal strategy is to split W into $\max(1, \lfloor n^* \rfloor)$ or $\lceil n^* \rceil$ same-size chunks, whichever leads to the smaller value. As stated, the value of y, hence of n^*, is independent of μ_d. ■

Proposition 4.3. *A first-order approximation for the optimal checkpointing period (that minimizes total execution time) is $T_{opt} \approx \sqrt{2C\mu_e} + C$. This value is identical to Young's formula, and does not depend on the value of μ_d.*

Proof. We use Proposition 4.2 and Taylor expansions when $z = y + 1 = \frac{\lambda_e W}{n^*}$ is small: from $ye^y = -e^{-\lambda_e C - 1}$, we derive $(z-1)e^z = -e^{-\lambda_e C}$. We have $(z-1)e^z \approx \frac{z^2}{2} - 1$, and $-e^{-\lambda_e C} \approx -1 + \lambda_e C$, hence $z^2 \approx 2\lambda_e C$. The period is

$$T_{\text{opt}} = \frac{W}{n^*} + C = \frac{z}{\lambda_e} + C \approx \sqrt{2C\mu_e} + C.$$

■

An intuitive explanation of the result is the following: error detection latency is paid for every error, and can be viewed as an additional downtime, which has no impact on the optimal period.

Arbitrary distributions

Here we extend the previous result to arbitrary distribution laws for X_e and X_d (of mean μ_e and μ_d respectively):

Proposition 4.4. *When $C \ll \mu_e$ and $\mu_d + D + R \ll \mu_e$, a first-order approximation for the optimal checkpointing period is $T_{opt} \approx \sqrt{2C\mu_e} + C$.*

Proof. Let $\mathcal{T}_{\text{base}}$ be the base time of the application without any overhead due to resilience techniques. First, assume a fault-free execution of the application: every period of length T, only $W = T - C$ units of work are executed, hence the time $\mathcal{T}_{\text{ff}}$ for a fault-free execution is $\mathcal{T}_{\text{ff}} = \frac{T}{W}\mathcal{T}_{\text{base}}$. Now, let $\mathcal{T}_{\text{final}}$ denote the expectation of the execution time with errors taken into account. In average, errors occur every μ_e time-units, and for each of them we lose $\mathcal{F}$ time-units, so there are $\frac{\mathcal{T}_{\text{final}}}{\mu_e}$ errors during the execution. Hence we derive that

$$\mathcal{T}_{\text{final}} = \mathcal{T}_{\text{ff}} + \frac{\mathcal{T}_{\text{final}}}{\mu_e}\mathcal{F}, \tag{4.6}$$

which we rewrite as

$$(1 - \text{WASTE})\mathcal{T}_{\text{final}} = \mathcal{T}_{\text{base}},$$
$$\text{with WASTE} = 1 - \left(1 - \frac{\mathcal{F}}{\mu_e}\right)\left(1 - \frac{C}{T}\right). \tag{4.7}$$

The waste is the fraction of time where nodes do not perform useful computations. Minimizing execution time is equivalent to minimizing the waste. In Equation (4.7), we identify the two sources of overhead: (i) the term $\text{WASTE}_{\text{FF}} = \frac{C}{T}$, which is the waste due to checkpointing in a fault-free execution, by construction of the algorithm; and (ii) the term $\text{WASTE}_{\text{Fail}} = \frac{\mathcal{F}}{\mu_e}$, which is the waste due to errors striking during execution. With these notations, we have

$$\text{WASTE} = \text{WASTE}_{\text{Fail}} + \text{WASTE}_{\text{FF}} - \text{WASTE}_{\text{Fail}}\text{WASTE}_{\text{FF}}. \tag{4.8}$$

As a sanity check, this is the same equation as Equation 1.13. There remains to determine the (expected) value of $\mathcal{F}$. Assuming at most one error per period, we lose $\mathcal{F} = \frac{T}{2} + \mu_d + D + R$ per error: $\frac{T}{2}$ for the average work lost before the error occurs, μ_d for detecting the error, and $D + R$ for downtime and recovery. Note that the assumption is valid only if $\mu_d + D + R \ll \mu_e$ and $T \ll \mu_e$. Plugging back this value into Equation (4.8), we obtain

$$\text{WASTE}(T) = \frac{T}{2\mu_e} + \frac{C(1 - \frac{D+R+\mu_d}{\mu_e})}{T} + \frac{D + R + \mu_d - \frac{C}{2}}{\mu_e} \tag{4.9}$$

which is minimal for

$$T_{\text{opt}} = \sqrt{2C(\mu_e - D - R - \mu_d)}. \tag{4.10}$$

We point out that this approach based on the waste leads to a different approximation formula for the optimal period, but $T_{\text{opt}} = \sqrt{2C(\mu_e - D - R - \mu_d)} \approx \sqrt{2C\mu_e} \approx \sqrt{2C\mu_e} + C$ up to second-order terms, when μ_e is large in front of the other parameters, including μ_d. For example, this approach does not allow us to handle the case $\mu_d = \mu_e$; in such a case, the optimal period is known only for exponential distributions, and is independent of μ_d, as proven by Proposition 4.2. ■

To summarize, the exact value of the optimal period is only known for exponential distributions and is provided by Proposition 4.2, while Young's formula can be used as a first-order approximation for any other distributions. Indeed, the optimal period is a trade-off between the overhead due to checkpointing ($\frac{C}{T}$) and the expected time lost per error ($\frac{T}{2\mu_e}$ plus some constant). Up to second-order terms, the waste is minimum when both factors are equal, which leads to Young's formula, and which remains valid regardless of error detection latencies.

4.2.2 Saving only k checkpoints

Lu, Zheng and Chien [82] propose a set of simulations to assess the overhead induced when keeping only the last k checkpoints (because of storage limitations). In the following, we derive an analytical approach to numerically solve the problem. The main difficulty is that when error detection latency is too large, it is impossible to recover from a valid checkpoint, and one must resume the execution from scratch. We consider this scenario as an *irrecoverable failure*, and we aim at guaranteeing that the risk of irrecoverable failure remains under a user-given threshold.

Assume that a job of total size W is partitioned into n chunks. What is the risk of irrecoverable failure during the execution of one chunk of size $\frac{W}{n}$ followed by its checkpoint? Let $T = \frac{W}{n} + C$ be the length of the period. Intuitively, the longer the period, the smaller the probability that an error that has just been detected took place more than k periods ago, thereby leading to an irrecoverable failure because the last valid checkpoint is not one of the k most recent ones.

Formally, there is an irrecoverable failure if: (i) there is an error detected during the period (probability $\mathbb{P}_{\text{fail}}$), and (ii) the sum of T_{lost}, the time elapsed since the last checkpoint, and of X_d, the error detection latency, exceeds kT (probability $\mathbb{P}_{\text{lat}}$). The value of $\mathbb{P}_{\text{fail}} = \mathbb{P}(X_e \leq T)$ is easy to compute from

the error distribution law. For instance with an exponential law, $\mathbb{P}_{\text{fail}} = 1 - e^{-\lambda_e T}$. As for $\mathbb{P}_{\text{lat}}$, we use an upper bound: $\mathbb{P}_{\text{lat}} = \mathbb{P}(T_{lost} + X_d \geq kT) \leq \mathbb{P}(T + X_d \geq kT) = \mathbb{P}(X_d \geq (k-1)T)$. The latter value is easy to compute from the error distribution law. For instance with an exponential law, $\mathbb{P}_{\text{lat}} = e^{-\lambda_d (k-1)T}$. Of course, if there is an error and the error detection latency does not exceed kT (probability (1-$\mathbb{P}_{\text{lat}}$)), we have to restart execution and face the same risk as before. Therefore, the probability of irrecoverable failure $\mathbb{P}_{\text{irrec}}$ can be recursively evaluated as $\mathbb{P}_{\text{irrec}} = \mathbb{P}_{\text{fail}}(\mathbb{P}_{\text{lat}} + (1 - \mathbb{P}_{\text{lat}})\mathbb{P}_{\text{irrec}})$, hence $\mathbb{P}_{\text{irrec}} = \frac{\mathbb{P}_{\text{fail}}\mathbb{P}_{\text{lat}}}{1 - \mathbb{P}_{\text{fail}}(1 - \mathbb{P}_{\text{lat}})}$. Now that we have computed $\mathbb{P}_{\text{irrec}}$, the probability of irrecoverable failure for a single chunk, we can compute the probability of irrecoverable failure for n chunks as $\mathbb{P}_{\text{risk}} = 1 - (1 - \mathbb{P}_{\text{irrec}})^n$. In full rigor, these expressions for $\mathbb{P}_{\text{irrec}}$ and $\mathbb{P}_{\text{risk}}$ are valid only for exponential distributions, because of the memoryless property, but they are a good approximation for arbitrary laws. Given a prescribed risk threshold ε, solving numerically the equation $\mathbb{P}_{\text{risk}} \leq \varepsilon$ leads to a lower bound $T_{\min}$ on T. Let T_{opt} be the optimal period given in Theorem 4.3 for an unbounded number of saved checkpoints. The best strategy is then to use the period $\max(T_{\min}, T_{\text{opt}})$ to minimize the waste while enforcing a risk below threshold.

In case of irrecoverable failure, we have to resume execution from the very beginning. The number of re-executions due to consecutive irrecoverable failures follows a geometric law of parameter $1 - \mathbb{P}_{\text{risk}}$, so that the expected number of executions until success is $\frac{1}{1 - \mathbb{P}_{\text{risk}}}$. We refer to Section 4.4.1 for an example of how to instantiate this model to compute the best period with a fixed number of checkpoints, under a prescribed risk threshold.

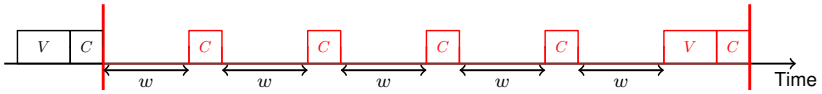
4.3 Coupling verification and checkpointing

In this section, we move to a more realistic model where silent errors are detected only when some verification mechanism (checksum, error correcting code, coherence tests, etc.) is executed. Our approach is agnostic of the nature of this verification mechanism. We aim at solving the following optimization problem: given the cost of checkpointing C, downtime D, recovery R, and verification V, what is the optimal strategy to minimize the expected waste as a function of the mean time between errors μ_e? Depending upon the relative costs of checkpointing and verifying, we may have more checkpoints than verifications, or the other way round. In both cases, we target a periodic pattern that repeats over time.

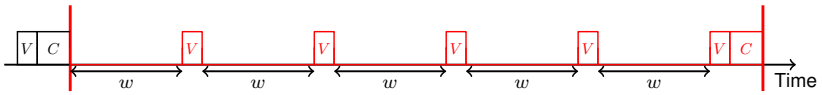
Consider first the scenario where the cost of a checkpoint is smaller than the cost of a verification: then the periodic pattern will include k checkpoints and 1 verification, where k is some parameter to determine. Figure 4.2(a) provides an illustration with $k = 5$. We assume that the verification is directly followed by the last checkpoint in the pattern, so as to save results just after they have been verified (and before they get corrupted). In this scenario, the objective is to determine the value of k that leads to the minimum platform waste. This problem is addressed in Section 4.3.1.

Because our approach is agnostic of the cost of the verification, we also envision scenarios where the cost of a checkpoint is higher than the cost of a verification. In such a framework, the periodic pattern will include k verifications and 1 checkpoint, where k is some parameter to determine. See Figure 4.2(b) for an illustration with $k = 5$. Again, the objective is to determine the value of k that leads to the minimum platform waste. This problem is addressed in Section 4.3.2.

We point out that combining verification and checkpointing guarantees that no irrecoverable failure will kill the application: the last checkpoint of any period pattern is always correct, because a verification always takes place right before this checkpoint is taken. If that verification reveals an error, we roll back until reaching a correct verification point, maybe up to the end of the previous pattern, but never further back, and re-execute the work. The amount of roll-back and re-execution depends upon the shape of the pattern, and we show how to compute it in Sections 4.3.1 and 4.3.2 below.



(a) five checkpoints for one verification



(b) five verifications for one checkpoint

Figure 4.2: Periodic pattern.

4.3.1 With k checkpoints and one verification

We use the same approach as in the proof of Proposition 4.4 and compute a first-order approximation of the waste (see Equations (4.7) and (4.8)). We compute the two sources of overhead: (i) WASTE_{FF}, the waste incurred in a fault-free execution, by construction of the algorithm, and (ii) $\text{WASTE}_{\text{Fail}}$, the waste due to errors striking during execution.

Let $\mathbb{S} = kw+kC+V$ be the length of the periodic pattern. We easily derive that $\text{WASTE}_{\text{FF}} = \frac{kC+V}{\mathbb{S}}$. As for $\text{WASTE}_{\text{Fail}}$, we still have $\text{WASTE}_{\text{Fail}} = \frac{D+\mathbb{E}(T_{lost})}{\mu_e}$. However, in this context, the time lost because of the error depends upon the location of this error within the periodic pattern, so we compute averaged values as follows. Recall (see Figure 4.2(a)) that checkpoint k is the one preceded by a verification. Here is the analysis when an error is detected during the verification that takes place in the pattern:

- If the error took place in the (last) segment k: we recover from checkpoint $k-1$, and verify it; we get a correct result because the error took place later on. Then we re-execute the last piece of work and redo the verification. The time that has been lost is $T_{lost}(k) = R+V+w+V$. (We assume that there is at most one error per pattern.)
- If the error took place in segment i, $2 \leq i \leq k-1$: we recover from checkpoint $k-1$, verify it, get a wrong result; we iterate, going back up to checkpoint $i-1$, verify it, and get a correct result because the error took place later on. Then we re-execute $k-i+1$ pieces of work and $k-i$ checkpoints, together with the last verification. We get $T_{lost}(i) = (k-i+1)(R+V+w)+(k-i)C+V$.
- If the error took place in (first) segment 1: this is almost the same as above, except that the first recovery at the beginning of the pattern need not be verified, because the verification was made just before the corresponding checkpoint at the end of the previous pattern. We have the same formula with $i=1$ but with one fewer verification: $T_{lost}(1) = k(R+w)+(k-1)(C+V)+V$.

Therefore, the formula for $\text{WASTE}_{\text{Fail}}$ writes

$$\text{WASTE}_{\text{Fail}} = \frac{D + \frac{1}{k}\sum_{i=1}^{k} T_{lost}(i)}{\mu_e}, \tag{4.11}$$

and (after some manipulation using a computer algebra system) the formula simplifies to

$$\text{WASTE}_{\text{Fail}} = \frac{1}{2k\mu_e}((R+V)k^2+(2D+R+2V+\mathbb{S}-2C)k+\mathbb{S}-3V) \tag{4.12}$$

Using $\text{WASTE}_{\text{FF}} = \frac{kC+V}{\mathbb{S}}$ and Equation (4.8), we compute the total waste and derive that $\text{WASTE} = a\mathbb{S}+b+\frac{c}{\mathbb{S}}$, where a, b, and c are some constants. The optimal value of $\mathbb{S}$ is $\mathbb{S}_{opt} = \sqrt{\frac{c}{a}}$, provided that this value is at least $kC+V$. We point out that this formula only is a first-order approximation. We have

assumed a single error per pattern. We have also assumed that errors did not occur during checkpoints following verifications. Now, once we have found $\text{WASTE}(\mathbb{S}_{opt})$, the value of the waste obtained for the optimal period $\mathbb{S}_{opt}$, we can minimize this quantity as a function of k, and numerically derive the optimal value k_{opt} that provides the best value (and hence the best platform usage).

Due to lack of space, computational details are available online [84], which is a Maple sheet that we have to instantiate the model. This Maple sheet is publicly available for users to experiment with their own parameters. We provide two example scenarios to illustrate the model in Section 4.4.3.

Finally, note that in order to minimize the waste, one could do a binary search in order to find the last checkpoint before the fault. Then we can upper-bound $T_{lost}(i)$ by $(k-i+1)w+\log(k)(R+V)+(k-i)C+V$, and Equation (4.12) becomes $\text{WASTE}_{\text{Fail}} = \frac{1}{2k\mu_e}((R+V)2k\log(k)+(2D+R+2V+\mathbb{S}-2C)k+\mathbb{S}-3V)$.

4.3.2 With k verifications and one checkpoint

We use a similar line of reasoning for this scenario and compute a first-order approximation of the waste for the case with k verifications and 1 checkpoint per pattern. The length of the periodic pattern is now $\mathbb{S} = kw+kV+C$. As before, for $1 \leq i \leq k$, let segment i denote the period of work before verification i, and assume (see Figure 4.2(b)) that verification k is preceded by a checkpoint. The analysis is somewhat simpler here.

If an error takes place in segment i, $1 \leq i \leq k$, we detect the error during verification i, we recover from the last checkpoint, and redo the first i segments and verifications: therefore $T_{lost}(i) = R+i(V+w)$. The formula for $\text{WASTE}_{\text{Fail}}$ is the same as in Equation (4.11) and (after some manipulation) we derive

$$\text{WASTE}_{\text{Fail}} = \frac{1}{2\mu_e}\left(D+R+\frac{k+1}{2k}(\mathbb{S}-C)\right). \tag{4.13}$$

Using $\text{WASTE}_{\text{FF}} = \frac{kV+C}{\mathbb{S}}$ and Equation (4.8), we proceed just as in Section 4.3.1 to compute the optimal value $\mathbb{S}_{opt}$ of the periodic pattern, and then the optimal value k_{opt} that minimizes the waste. Details are available within the Maple sheet [84].

4.4 Evaluation

This section provides some examples for instantiating the various models. We aimed at choosing realistic parameters in the context of future Exascale platforms, but we had to restrict to a limited set of scenarios, which do not intend to cover the whole spectrum of possible parameters. The Maple sheet [84] is available to explore other scenarios.

4.4.1 Best period with k checkpoints under a given risk threshold

We first evaluate $\mathbb{P}_{\text{risk}}$, the risk of irrecoverable failure, as defined in Section 4.2.2. Figures 4.3 and 4.4 present, for different scenarios, the probability $\mathbb{P}_{\text{risk}}$ as a function of the checkpointing period T on the left. On the right, the figures present the corresponding waste with k checkpoints and in the absence of irrecoverable failures. This waste can be computed following the same reasoning as in Equation (4.9). For each figure, the left diagram represents the risk implied by a given period T, showing the value T_{opt} of the optimal checkpoint interval (optimal with respect to waste minimization and in the absence of irrecoverable failures, see Equation (4.10)) as a blue vertical line. The right diagram on the figure represents the corresponding waste, highlighting the trade-off between an increased irrecoverable-failure-free

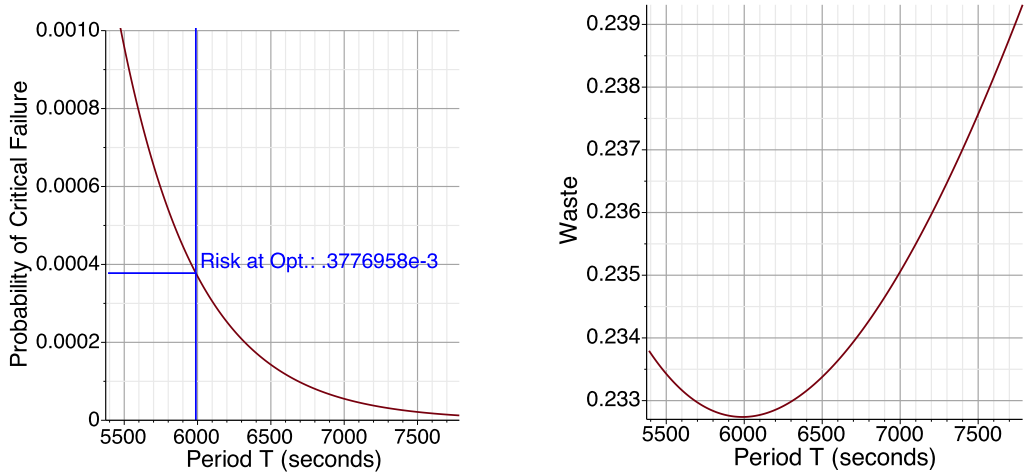


Figure 4.3: Risk of irrecoverable failure as a function of the checkpointing period, and corresponding waste. ($k = 3$, $\lambda_e = \frac{10^5}{100y}$, $\lambda_d = 30\lambda_e$, $w = 10d$, $C = R = 600s$, and $D = 0s$.)

waste and a reduced risk. As stated in Section 4.2.2, it does not make sense to select a value for T lower than T_{opt}, since the waste would be increased, for an increased risk.

Figure 4.3 considers a machine consisting of 10^5 components, and a component MTBF of 100 years. This component MTBF corresponds to the optimistic assumption on the reliability of computers made in the literature [26, 39]. The platform MTBF μ_e is thus $100 \times 365 \times 24/100,000 \approx 8.76$ hours. The times to checkpoint and recover (10 min) correspond to reasonable mean values for systems at this size [17, 43]. At this scale, process rejuvenation is small, and we set the downtime to 0s. For these average values to have a meaning, we consider a run that is long enough (10 days of work), and in order to illustrate the trade-off, we take a rather low (but reasonable) value $k = 3$ of intervals, and a mean time error detection μ_d significantly smaller (30 times) than the MTBF μ_e itself.

With these parameters, T_{opt} is around 100 minutes, and the risk of irrecoverable failure at this checkpoint interval can be evaluated at $1/2617 \approx 38 \cdot 10^{-5}$, inducing an irrecoverable-failure-free waste of 23.45%. To reduce the risk to 10^{-4}, a T_{min} of 8000 seconds is sufficient, increasing the waste by only 0.6%. In this case, the benefit of fixing the period to $\max(T_{\text{opt}}, T_{\text{min}})$ is obvious. Naturally, keeping a bigger amount of checkpoints (increasing k) would also reduce the risk, at constant waste, if memory can be afforded.

We also consider in Figure 4.4 a more optimistic scenario where the checkpointing technology and availability of resources is increased by a factor 10: the time to checkpoint, recover, and allocate new computing resources is divided by 10 compared to the previous scenario. Other parameters are kept similar. One can observe that T_{opt} is largely reduced (down to less than 35 minutes between checkpoints), as well as the optimal irrecoverable-failure-free waste (9.55%). This is unsurprising, and mostly due to the reduction of failure-free waste implied by the reduction of checkpointing time. But because the period between checkpoints becomes smaller, while the latency to detect an error is unchanged (μ_d is still 30 times smaller than μ_e), the risk that an error happens at the interval i but is detected after interval $i + k$ is increased. Thus, the risk climbs to $1/2$, an unacceptable value. To reduce the risk to 10^{-4} as previously, it becomes necessary to consider a T_{min} of 6650 seconds, which implies an irrecoverable-failure-free waste of 15%, significantly higher than the optimal one, which is below 10%, but still much lower than the 24% when checkpoint and availability costs are 10 times higher.

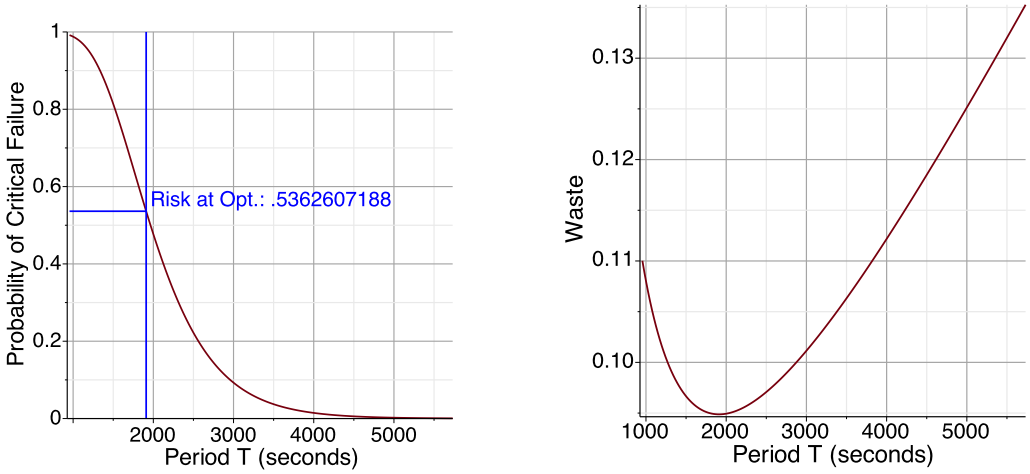


Figure 4.4: Risk of irrecoverable failure as a function of the checkpointing period, and corresponding waste. ($k = 3$, $\lambda_e = \frac{10^5}{100y}$, $\lambda_d = 30\lambda_e$, $w = 10d$, $C = R = 60s$, and $D = 0s$.)

4.4.2 Periodic pattern with k verifications and one checkpoint

We now focus on the waste induced by the different ways of coupling periodic verification and checkpointing. We first consider the case of a periodic pattern with more verifications than checkpoints: every k verifications of the current state of the application, a checkpoint is taken. The duration of the work interval $\mathbb{S}$, between two verifications in this case, is optimized to minimize the waste. We consider two scenarios. For each scenario, we represent two diagrams: the left diagram shows the waste as a function of k for a given verification cost V, and the right diagram shows the waste as a function of k and V using a 3D surface representation.

In the first scenario, we consider the same setup as above in Section 4.4.1. The waste is computed in its general form, so we do not need to define the duration of the work. As represented in Figure 4.5, for a given verification cost, the waste can be optimized by making more than one verification. When $k > 1$, there are intermediate verifications that can enable the detection of an error before a periodic pattern (of length $\mathbb{S}$) is completed, hence, that can reduce the time lost due to an error. However, introducing too many verifications induces an overhead that eventually dominates the waste. The 3D surface shows that the waste reduction is significant when increasing the number of verifications, until the optimal number is reached. Then, the waste starts to increase again slowly. Intuitively, the lower the cost for V, the higher the optimal value for k.

When considering the second scenario (Figure 4.6), with an improved checkpointing and availability setup, the same conclusions can be reached, with an absolute value of the waste greatly diminished. Since forced verifications allow to detect the occurrence of errors at a controllable rate (depending on $\mathbb{S}$ and k), the risk of non-recoverable errors is nonexistent in this case, and the waste can be greatly diminished, with very few checkpoints taken and kept during the execution.

4.4.3 Periodic pattern with k checkpoints and one verification

The last set of experiments considers the opposite case of periodic patterns: checkpoints are taken more often than verifications. Every k checkpoints, a verification of the data consistency is done. Intuitively, this could be useful if the cost of verification is large compared to the cost of checkpointing itself. In

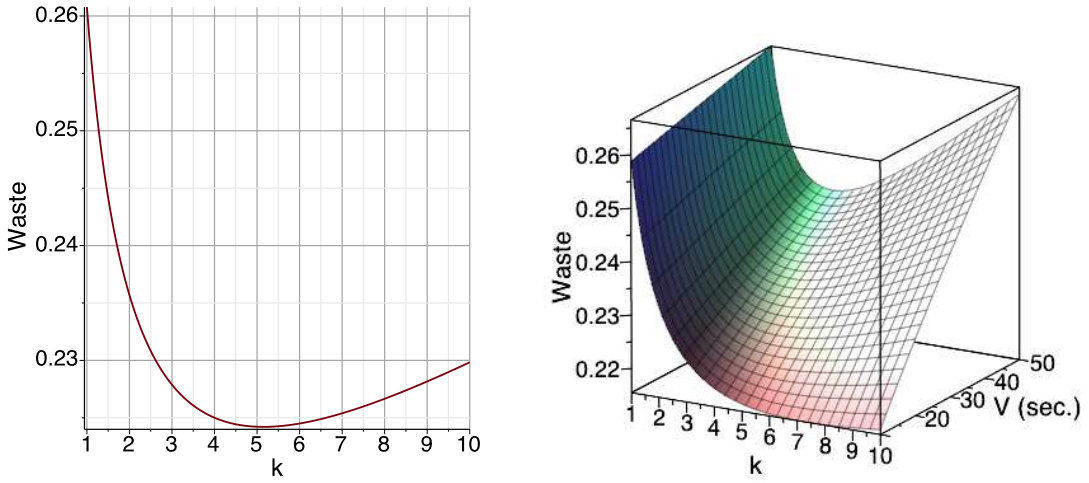


Figure 4.5: Case with k verifications, and one checkpoint per periodic pattern. Waste as function of k, and potentially of V, using the optimal period. ($V = 20s, C = R = 600s, D = 0s$, and $\mu = \frac{10y}{10^5}$.)

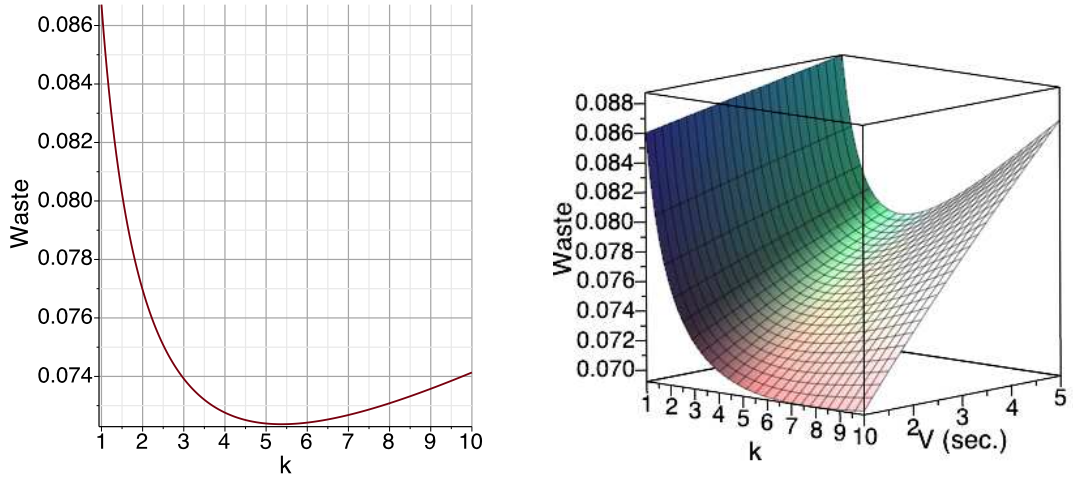


Figure 4.6: Case with k verifications, and one checkpoint per periodic pattern. Waste as function of k, and potentially of V, using the optimal period. ($V = 2s, C = R = 60s, D = 0s$, and $\mu = \frac{10y}{10^5}$.)

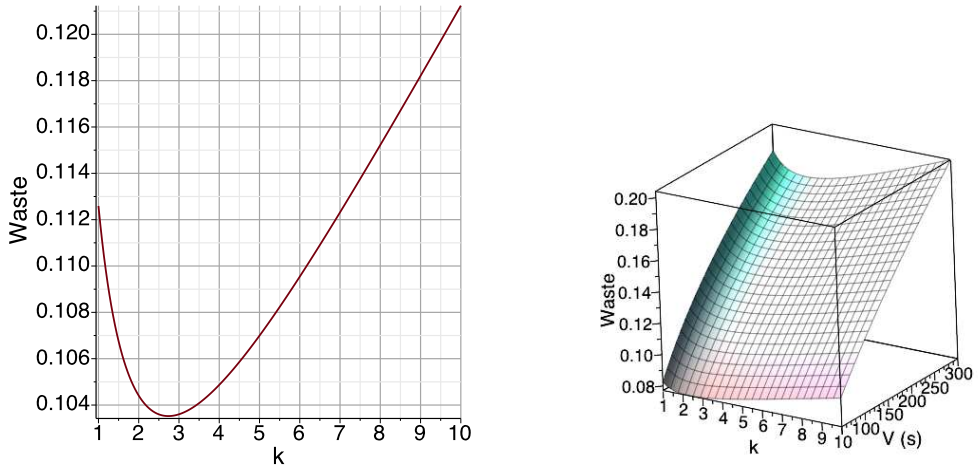


Figure 4.7: Case with k checkpoints, and one verification per periodic pattern. Waste as function of k, and potentially of V, using the optimal period. ($V = 100s, C = R = 6s, D = 0s$, and $\mu = \frac{10y}{10^5}$.)

that case, when rolling back after an error is discovered, each checkpoint that was not validated before is validated at rollback time, potentially invalidating up to $k - 1$ checkpoints.

Because this pattern has potential only when the cost of checkpoint is much lower than the cost of verification, we considered the case of a greatly improved checkpoint / availability setup: the checkpoint and recovery times are only 6 seconds in Figure 4.7. One can observe that in this extreme case, it can still make sense to consider multiple checkpoints between two verifications (when $V = 100$ seconds, a verification is done only every 3 checkpoints optimally); however the 3D surface demonstrates that the waste is still dominated by the cost of the verification, and little improvement can be achieved by taking the optimal value for k. The cost of verification must be incurred when rolling back, and this shows on the overall performance if the verification is costly.

This is illustrated even more clearly with Figure 4.8, where the checkpoint costs and machine availability are set to the second scenario of Sections 4.4.1 and 4.4.2. As soon as the checkpoint cost is not negligible compared to the verification cost (only 5 times smaller in this case), it is more efficient to validate every other checkpoint than to validate only after $k > 2$ checkpoints. The 3D surface shows that this holds true for rather large values of V.

All the rollback / recovery techniques that we have evaluated above, using various parameters for the different costs, and stressing the different approaches to their limits, expose a waste that remains, in the vast majority of the cases, largely below 66%. This is noticeable, because the traditional hardware based technique, which relies on triple modular redundancy and voting [83], mechanically presents a "waste" of resources that is at least equal to 66% (two-thirds of resources are wasted, and neglecting the cost of voting).

4.5 Conclusion

In this chapter, we have revisited traditional checkpointing and rollback recovery strategies. Rather than considering fail-stop failures, we focus on silent data corruption errors. Such latent errors cannot be neglected anymore in High Performance Computing, in particular in sensitive and high precision simulations. The core difference with fail-stop failures is that error detection is not immediate.

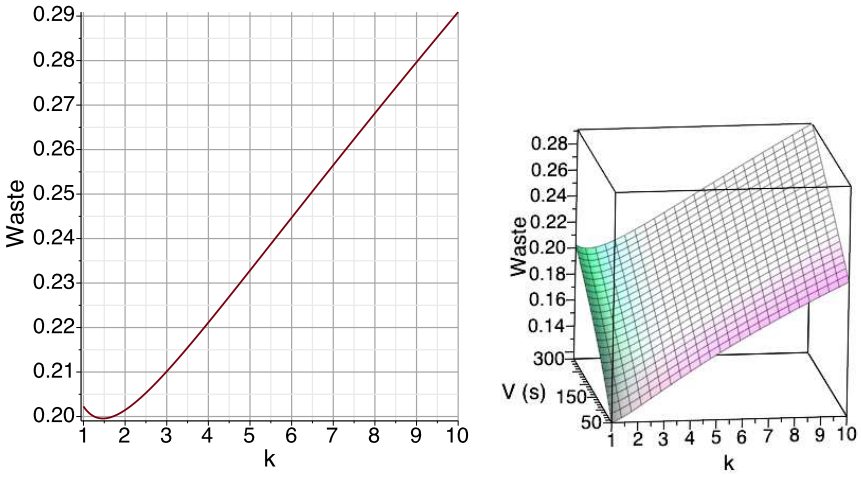


Figure 4.8: Case with k checkpoints, and one verification per periodic pattern. Waste as function of k, and potentially of V, using the optimal period. ($V = 300s, C = R = 60s, D = 0s$, and $\mu = \frac{10y}{10^5}$.)

We discuss and analyze two models. In the first model, errors are detected after some delay following a probability distribution (typically, an exponential distribution). We compute the optimal checkpointing period in order to minimize the waste when all checkpoints can be kept in memory, and we show that this period does not depend on the distribution of detection times. In practice, only a few checkpoints can be kept in memory, and hence it may happen that an error was detected after the last correct checkpoint was removed from storage. We derive a minimum value of the period to guarantee, within a risk threshold, that at least one valid checkpoint remains when a latent error is detected.

A more realistic model assumes that errors are detected through some verification mechanism. Periodically, one checks whether the current status is meaningful or not, and then eventually detects a latent error. We discuss both the case where the periodic pattern includes k checkpoints for one verification (large cost of verification), and the opposite case with k verifications for one checkpoint (inexpensive cost for verification). We express a formula for the waste in both cases, and, from these formulas, we derive the optimal period.

The various models are instantiated with realistic parameters, and the evaluation results clearly corroborate the theoretical analysis. For the first model, with detection times, the tradeoff between waste and risk of irrecoverable error clearly appears, hence showing that a period larger than the one minimizing the irrecoverable-failure-free waste should often be chosen to achieve an acceptable risk. The advantage of the second model is that there are no irrecoverable failures (within each period, there is a verification followed by a checkpoint, hence ensuring a valid checkpoint). We compute the optimal pattern of checkpoints and verifications per period, as a function of their respective cost, to minimize the waste. The pattern with more checkpoints than verification turns out to be usable only when the cost of checkpoint is much lower than the cost of verification, and the conclusion is that it is often more efficient to verify the result every other checkpoint.

Overall, we provide a thorough analysis of checkpointing models for latent errors, both analyzing the models analytically, and evaluating them through different scenarios. A future research direction would be to study more general scenarios of multiple checkpointing, for instance by keeping not the consecutive k last checkpoints in the first model, but rather some older checkpoints to decrease the risk. In the second model, more verification/checkpoint combinations could be studied, while we focused on cases with an integer number of checkpoints per verification (or the converse).

Part II

Reliable and energy-aware schedules

Chapter 5

Introduction to energy-efficient scheduling

The *energy consumption* of computational platforms has recently become a critical problem, not only for economic and environmental reasons [88], but also for building the future generation of Exascale system. The Defense Advanced Research Projects Agency (DARPA) capped to a maximum of 20 MW the energetic consumption of future Exascale systems [12]. As an example, the Earth Simulator requires about 12 MW (Mega Watts) of peak power, and current Petaflop systems require more than 17 MW of power. The Green500 list (`www.green500.org`) provides rankings of the most energy-efficient supercomputers in the world, therefore raising even more awareness about power consumption. For their November 2013 ranking, it is the first time that a supercomputer has broken the 4 Gigaflops/W barrier. That supercomputer is the Tsubame-KFC at the Tokyo Institute of Technology, ranked 311 amongst the top 500 post powerful supercomputer (only 2720 cores). In the 15 most energy efficient supercomputers, there is only one amongst the top 10 most efficient supercomputers, PizDaint, at the Swiss National Supercomputing Centre, currently ranked 4^{th} on the Green500 list, with 3 Gigaflops/W, and ranked 6^{th} on the TOP500 list, with 115984 nodes. In the mean time, the top3 supercomputers (according to the TOP500 list), reach a 2 Gigaflops/W energetic efficiency. To reach the 20 MW limit, Exascale systems will need to have an energetic efficiency of 50 Gigaflops/W. For the first time, with the November 2013 evaluation of supercomputers, the extrapolation (assuming that the Tsubame-KFC performance can be maintained on an Exascale system) of the energy performance of an Exascale system has dropped below 300 MW!

On the economical side, at $200 per MW.Hour, peak operation of a Petaflop machine may thus cost between $3,400 to $10,000 per hour [49], that is $30 million per year. Current estimates state that cooling costs $1 to $3 per watt of heat dissipated [114]. This is just one of the many economical reasons why energy-aware scheduling has proved to be an important issue in the past decade, even without considering battery-powered systems such as laptops and embedded systems.

To help reduce energy dissipation, many efforts have been devoted to improving the physical capacitance or switching activity of transistors. The consumed power in a CPU, when powered on, is divided into a static part $\mathcal{P}_{\text{Static}}$ and a dynamic part $\mathcal{P}_{\text{Cal}}$. The static part is the cost for a processor to be on, whereas the dynamic part is an additional cost for the computations, according to the speed at which the processor is running. More precisely, we have: $\mathcal{P}_{\text{Static}} = VI_{leak}$ and $\mathcal{P}_{\text{Cal}} = aCV_{DD}^2 f$, where:

- a is the switching activity: the chip is composed of transistors that switch during a computation;
- C is the physical capacitance;
- f is the clock frequency (or speed), the number of operations per second;
- V_{DD} is the supply voltage, which is a function of the clock frequency;

- I_{leak} is the leakage current, due to the nature of the transistors.

A lot of research has been done with the aim to reduce power consumption through hardware modifications, and tries to act on all those factors. Transistors have been improved, in order to reduce a and C. The temperature plays an important role, through the leakage current. When the temperature is rising, the leakage current is increasing, thus the dissipated power is higher, which leads to an elevation of the temperature. This vicious circle must be avoided, thanks to effective cooling systems. But the saved power in the processor must be greater than the power needed to make the cooling system work.

The most important breakthrough is the advent of the Dynamic Voltage and Frequency Scaling (DVFS), which is enabled on almost all recent processors. The new generation is indeed able to run at different speeds and voltages, those parameters being set up by the user. Several models exist to describe the DVFS technique. It is commonly assumed that the voltage is proportional to the frequency, implying a dynamic power in the cube of the frequency [62, 65, 101, 28, 6, 31]. Faster speeds allow for a faster execution, but they also lead to a much higher (supra-linear) power consumption. In this part of the thesis, when considering the DVFS technique, we only refer to what is called CONTINUOUS frequencies (or speeds). Processors can have arbitrary speeds, and can vary them continuously: while this model is unrealistic, it is theoretically appealing [8]. Many papers (see Section 5.3) considered discrete speeds, a more realistic model. We have ourselves extended most of this work to discrete speed models [J1, C2, RR3].

Energy-aware scheduling aims at minimizing the energy consumed during the execution of the target application. As an introduction to reliable and energy efficient algorithms, we present in Section 5.1 some work where reliability is not a constraint. Then, in Section 5.2 we introduce how this energy-aware scheduling impacts reliability. We conclude this chapter by a survey of related literature in Section 5.3.

5.1 Reclaiming the energy of a schedule: Models and algorithms

In this section, we investigate energy-aware scheduling strategies for executing a task graph on a set of processors. The main originality is that we assume that the mapping of the task graph is given, say by an ordered list of tasks to execute on each processor. There are many situations in which this problem is important, such as optimizing for legacy applications, or accounting for affinities between tasks and resources, or even when tasks are pre-allocated [103], for example for security reasons. In such situations, assume that a list-schedule has been computed for the task graph, and that its execution time should not exceed a deadline D. We do not have the freedom to change the assignment of a given task, but we can change its speed to reduce energy consumption, provided that the deadline D is not exceeded after the speed change. Rather than using a local approach such as backfilling [118, 100], which only reclaims gaps in the schedule, we consider the problem as a whole, and we assess the impact of speed scaling on its complexity. More precisely, we investigate the CONTINUOUS speed model: processors can have arbitrary speeds, and can vary them continuously. This model is unrealistic (any possible value of the speed, say $\sqrt{e^{\pi}}$, cannot be obtained) but it is theoretically appealing [8].

We present some polynomial time optimal algorithms for special graph structures, such as trees and series-parallel graphs, and we cast the problem into a geometric programming problem [22] for general DAGs.

The section is organized as follows. We start by providing the formal description of the framework in Section 5.1.1. Then in Section 5.1.2, we provide analytical formulas, and the formulation into a convex optimization problem. Finally, we conclude in Section 5.1.3.

5.1.1 Framework

Consider an application task graph $\mathcal{G} = (V, \mathcal{E})$, where the vertex set $V = \{T_1, \ldots, T_n\}$ is the set of n tasks, and the edge set $\mathcal{E}$ corresponds to the precedence constraints between tasks. Task T_i has a weight w_i for $1 \leq i \leq n$. We assume that the tasks in $\mathcal{G}$ have been allocated onto a parallel platform made up of identical processors. Each processor has an interval $[f_{\text{min}}, f_{\text{max}}]$ of available speeds. We consider in this section the special case where $f_{\text{min}} = 0$ (in the next chapters we consider the general case) to simplify the proofs for this introductory section. However, the proofs would also work with a minimum speed different from zero.

We define the *execution graph* generated by this allocation as the graph $G = (V, E)$, with the following augmented set of edges:

- $\mathcal{E} \subseteq \tilde{\mathcal{E}}$: if an edge exists in the precedence graph, it also exists in the execution graph;
- if T_1 and T_2 are executed successively, in this order, on the same processor, then $(T_1, T_2) \in \tilde{\mathcal{E}}$ is added as a precedence constraint.

The goal is to minimize the energy consumed during the execution while enforcing a deadline D on the execution time. We formalize the different models in the simpler case where each task is executed at constant speed. This strategy is optimal for the CONTINUOUS model (see Lemma 5.1).

Makespan

The makespan of a schedule is its total *execution time*. The first task is scheduled at time 0, so that the makespan of a schedule is simply the maximum time at which one of the processors finishes its computations. We consider a deadline bound D, which is a constraint on the makespan.

Let $\mathcal{E}xe(w_i, f_i)$ be the execution time of a task T_i of weight w_i at speed f_i. We assume that the cache size is adapted to the application, therefore ensuring that the execution time is linearly related to the frequency [85]: $\mathcal{E}xe(w_i, f_i) = \frac{w_i}{f_i}$.

Energy model

If task T_i is executed at speed f_i , the consumed energy is

$$E_i = E_i(f_i) = \mathcal{E}xe(w_i, f_i) \times f_i^3, \tag{5.1}$$

which corresponds to the dynamic part of the classical energy models of the literature [65, 101, 28, 6, 31]. Recall that we do not take static energy into account, because all processors are up and alive during the whole execution.

The total energy consumed by the schedule is the sum over all tasks of the energy consumed for each task:

$$\text{ENERGY}: \quad E = \sum_{i=1}^{n} E_i \ , \tag{5.2}$$

where E_i is defined by Equation (5.1).

Optimization problem

For each task $T_i \in V$, b_i is the starting time of its execution, d_i is the duration of its execution, and f_i is the speed at which it is executed. We obtain the following formulation of the MINENERGY(G, D)

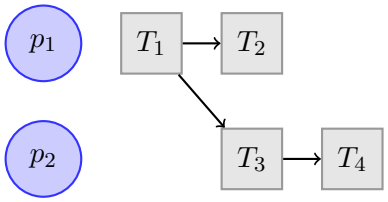


Figure 5.1: Execution graph for the example.

problem, given an execution graph $G = (V, E)$ and a deadline D; the f_i values are variables, whose values are constrained by the energy model.

$$\begin{array}{lll} \text{Minimize} & & \sum_{i=1}^{n} f_i^3 \times d_i \\ \text{subject to} & \text{(i)} & w_i = f_i \times d_i \text{ for each task } T_i \in V, \\ & \text{(ii)} & b_i + d_i \leq b_j \text{ for each edge } (T_i, T_j) \in \tilde{\mathcal{E}}, \\ & \text{(iii)} & b_i + d_i \leq D \text{ for each task } T_i \in V, \\ & \text{(iv)} & b_i \geq 0 \text{ for each task } T_i \in V. \end{array} \tag{5.3}$$

Constraint (i) states that the whole task can be executed in time d_i using speed f_i. Constraint (ii) accounts for all dependencies, and constraint (iii) ensures that the execution time does not exceed the deadline D. Finally, constraint (iv) enforces that starting times are non-negative. The energy consumed throughout the execution is the objective function. It is the sum, for each task, of the energy consumed by this task. Note that $d_i = \mathcal{E}xe(w_i, f_i) = w_i/f_i$, and therefore the objective function can also be expressed as $\sum_{i=1}^{n} f_i^2 \times w_i$.

Recall that there is a maximum speed that cannot be exceeded, denoted $f_{\max}$. We point out that there is a solution to the minimization problem if and only if there is a solution with $f_i = f_{\max}$ for all $1 \leq i \leq n$. Such a solution would correspond to executing each task as early as possible (according to constraints (ii) and (iv)) and as fast as possible. The optimal solution then slows down tasks to save as much energy as possible, while enforcing the deadline constraint. There is no guarantee on the uniqueness of the solution, since it may be possible to modify the beginning time of a task without affecting the energy consumption, if some of the constraints (ii) are not tight.

Example

Consider an application with four tasks of weights $w_1 = 3$, $w_2 = 2$, $w_3 = 1$ and $w_4 = 2$, and one precedence constraint $T_1 \rightarrow T_3$. We assume that T_1 and T_2 are allocated, in this order, onto processor P_1, while T_3 and T_4 are allocated, in this order, on processor P_2. The resulting execution graph G is given in Figure 5.1, with two precedence constraints added to the initial task graph. The deadline on the execution time is $D = 1.5$.

We set the maximum speed to $f_{\max} = 6$. We aim at finding the optimal execution speed f_i for each task T_i ($1 \leq i \leq 4$), i.e., the values of f_i that minimize the energy consumption.

With the CONTINUOUS model, the optimal speeds are non rational values, and we obtain

$$f_1 = \frac{2}{3}(3 + 35^{1/3}) \simeq 4.18; \quad f_2 = f_1 \times \frac{2}{35^{1/3}} \simeq 2.56; \quad f_3 = f_4 = f_1 \times \frac{3}{35^{1/3}} \simeq 3.83.$$

Note that all speeds are lower than the maximum $f_{\max}$. These values are obtained thanks to the formulas derived in Section 5.1.2. The energy consumption is then $E_{opt}^{(c)} = \sum_{i=1}^{4} w_i \times f_i^2 = 3.f_1^2 +$

$2.f_2^2 + 3.f_3^2 \simeq 109.6$. The execution time is $\frac{w_1}{f_1} + \max\left(\frac{w_2}{f_2}, \frac{w_3+w_4}{f_3}\right)$, and with this solution, it is equal to the deadline D (actually, both processors reach the deadline, otherwise we could slow down the execution of one task).

We conclude the study of this simple example with a short discussion on the energy savings that can be achieved. Executing the four tasks at maximum speed leads to consuming an energy $E_{max} = 8 \times 6^2 = 288$. Such an execution completes within a delay $D = 1$. We clearly see the trade-off between execution time and energy consumption here, since we gain almost two third of the energy by slowing down the execution from $D = 1$ to $D = 1.5$. Note that with $D = 1$, we can still slow down task T_2 to speed 4, and still gain a little over the brute force solution. Hence, even such a toy example allows us to illustrate the benefits of energy-aware schedules. Obviously, with larger examples, the energy savings will be even more dramatic, depending upon the range of available speeds and the tightness of the execution deadline. In fact, the maximal energy gain that can be achieved is not bounded: when executing each task as slow as possible (instead of as fast as possible), we gain $\left(\frac{f_{\text{max}}}{f_{\text{min}}}\right)^2 W_{\text{total}}$, where W_{total} is the sum of all task weights, and this quantity can be arbitrarily large.

5.1.2 Theoretical results

With the CONTINUOUS model, processor speeds can take any value between 0 and f_{max}. First we prove that, with this model, the processors do not change their speed during the execution of a task. Then, we derive the optimal speed values for special execution graph structures, expressed as closed form algebraic formulas, and we show that these values may be irrational (as already illustrated in the example in Section 5.1.1). Finally, we formulate the problem for general DAGs as a convex optimization program.

Preliminary lemma

Lemma 5.1 (constant speed per task)**.** *In all optimal solution with the* CONTINUOUS *model, each task is executed at constant speed, i.e., a processor does not change its speed during the execution of a task.*

Proof. Suppose that in the optimal solution, there is a task whose speed changes during the execution. Consider the first time-step at which the change occurs: the computation begins at speed f from time t to time t', and then continues at speed f' until time t''. The total energy consumption for this task in the time interval $[t; t'']$ is $E = (t' - t) \times f^3 + (t'' - t') \times (f')^3$. Moreover, the amount of work done for this task is $W = (t' - t) \times f + (t'' - t') \times f'$.

If we run the task during the whole interval $[t; t'']$ at constant speed $W/(t'' - t)$, the same amount of work is done within the same time. However, the energy consumption during this interval of time is now $E' = (t'' - t) \times (W/(t'' - t))^3$. By convexity of the function $x \mapsto x^3$, we obtain $E' < E$ since $t < t' < t''$. This contradicts the hypothesis of optimality of the first solution, which concludes the proof. ■

Special execution graphs

Independent tasks. Consider the problem of minimizing the energy of n independent tasks (i.e., each task is mapped onto a distinct processor, and there are no precedence constraints in the execution graph), while enforcing a deadline D.

Proposition 5.1 (independent tasks)**.** *When G is composed of independent tasks $\{T_1, \ldots, T_n\}$, the optimal solution to* MINENERGY(G, D) *is obtained when each task T_i ($1 \leq i \leq n$) is computed at speed $f_i = \frac{w_i}{D}$. If there is a task T_i such that $f_i > f_{max}$, then the problem has no solution.*

Proof. For task T_i, the speed f_i corresponds to the slowest speed at which the processor can execute the task, so that the deadline is not exceeded. If $f_i > f_{\max}$, the corresponding processor will never be able to complete its execution before the deadline, therefore there is no solution. To conclude the proof, we note that any other solution would meet the deadline constraint, and therefore the f_i's should be such that $\frac{w_i}{f_i} \leq D$, which means that $f_i \geq \frac{w_i}{D}$. These values would all be higher than the f_i's of the optimal solution, and hence would lead to a higher energy consumption. Therefore, this solution is optimal. ■

Linear chain of tasks. This case corresponds for instance to n independent tasks $\{T_1, \ldots, T_n\}$ executed onto a single processor. The execution graph is then a linear chain (order of execution of the tasks), with $T_i \to T_{i+1}$, for $1 \leq i < n$.

Proposition 5.2 (linear chain)**.** *When G is a linear chain of tasks, the optimal solution to* MINENERGY(G, D) *is obtained when each task is executed at speed $f = \frac{W}{D}$, with $W = \sum_{i=1}^{n} w_i$. If $f > f_{max}$, then there is no solution.*

Proof. Suppose that in the optimal solution, tasks T_i and T_j are such that $f_i < f_j$. The total energy consumption is E_{opt}. We define f such that the execution of both tasks running at speed f takes the same amount of time than in the optimal solution, i.e., $(w_i + w_j)/s = w_i/f_i + w_j/f_j$: $f = \frac{(w_i+w_j)}{w_i f_j + w_j f_i} \times f_i f_j$. Note that $f_i < f < f_j$ (it is the barycenter of two points with positive mass).

We consider a solution such that the speed of task T_k, for $1 \leq k \leq n$, with $k \neq i$ and $k \neq j$, is the same as in the optimal solution, and the speed of tasks T_i and T_j is f. By definition of f, the execution time has not been modified. The energy consumption of this solution is E, where $E_{opt} - E = w_i f_i^2 + w_j f_j^2 - (w_i + w_j) f^2$, i.e., the difference of energy with the optimal solution is only impacted by tasks T_i and T_j, for which the speed has been modified. By convexity of the function $x \mapsto x^2$, we obtain $E_{opt} > E$, which contradicts its optimality. Therefore, in the optimal solution, all tasks have the same execution speed. Moreover, the energy consumption is minimized when the speed is as low as possible, while the deadline is not exceeded. Therefore, the execution speed of all tasks is $f = W/D$. ■

Corollary 5.1. *A linear chain with n tasks is equivalent to a single task of weight $W = \sum_{i=1}^{n} w_i$.*

Indeed, in the optimal solution, the n tasks are executed at the same speed, and they can be replaced by a single task of weight W, which is executed at the same speed and consumes the same amount of energy.

Fork and join graphs. Let $V = \{T_1, \ldots, T_n\}$. We consider either a fork graph $G = (V \cup \{T_0\}, E)$, with $E = \{(T_0, T_i), T_i \in V\}$, or a join graph $G = (V \cup \{T_0\}, E)$, with $E = \{(T_i, T_0), T_i \in V\}$. T_0 is either the source of the fork or the sink of the join.

Theorem 5.1 (fork and join graphs)**.** *When G is a fork (resp. join) execution graph with $n + 1$ tasks $T_0, T_1, \ldots, T_n$, the optimal solution to* MINENERGY(G, D) *is the following:*

- *the execution speed of the source (resp. sink) T_0 is $f_0 = \dfrac{\left(\sum_{i=1}^{n} w_i^3\right)^{\frac{1}{3}} + w_0}{D}$;*
- *for the other tasks T_i, $1 \leq i \leq n$, we have $f_i = f_0 \times \dfrac{w_i}{\left(\sum_{i=1}^{n} w_i^3\right)^{\frac{1}{3}}}$ if $f_0 \leq f_{max}$.*

Otherwise, T_0 should be executed at speed $f_0 = f_{max}$, and the other speeds are $f_i = \frac{w_i}{D'}$, with $D' = D - \frac{w_0}{f_{max}}$, if they do not exceed f_{max} (Proposition 5.1 for independent tasks). Otherwise there is no solution.

If no speed exceeds f_{max}, the corresponding energy consumption is

$$\mathbf{minE}(G, D) = \frac{\left(\left(\sum_{i=1}^{n} w_i^3\right)^{\frac{1}{3}} + w_0\right)^3}{D^2} .$$

Proof. Let $t_0 = \frac{w_0}{f_0}$. Then, the source or the sink requires a time t_0 for execution. For $1 \leq i \leq n$, task T_i must be executed within a time $D - t_0$ so that the deadline is respected. Given t_0, we can compute the speed f_i for task T_i using Theorem 5.1, since the tasks are independent: $f_i = \frac{w_i}{D-t_0} = w_i \cdot \frac{f_0}{f_0 D - w_0}$. The objective is therefore to minimize $\sum_{i=0}^{n} w_i f_i^2$, which is a function of f_0:

$$\sum_{i=0}^{n} w_i f_i^2 = w_0 f_0^2 + \sum_{i=1}^{n} w_i^3 \cdot \frac{f_0^2}{(f_0 D - w_0)^2} = f_0^2 \left(w_0 + \frac{\sum_{i=1}^{n} w_i^3}{(f_0 D - w_0)^2}\right) = f(f_0).$$

Let $W_3 = \sum_{i=1}^{n} w_i^3$. In order to find the value of f_0 that minimizes this function, we study the function $f(x)$, for $x > 0$. $f'(x) = 2x\left(w_0 + \frac{W_3}{(xD-w_0)^2}\right) - 2D \cdot x^2 \cdot \frac{W_3}{(xD-w_0)^3}$, and therefore $f'(x) = 0$ for $x = (W_3^{\frac{1}{3}} + w_0)/D$. We conclude that the optimal speed for task T_0 is $f_0 = \frac{\left(\sum_{i=1}^{n} w_i^3\right)^{\frac{1}{3}} + w_0}{D}$, if $f_0 \leq f_{\max}$. Otherwise, T_0 should be executed at the maximum speed $f_0 = f_{\max}$, since it is the bottleneck task. In any case, for $1 \leq i \leq n$, the optimal speed for task T_i is $f_i = w_i \frac{f_0}{f_0 D - w_0}$.

Finally, we compute the exact expression of $\mathbf{minE}(G, D) = f(f_0)$, when $f_0 \leq f_{\max}$:

$$f(f_0) = f_0^2 \left(w_0 + \frac{W_3}{(f_0 D - w_0)^2}\right) = \left(\frac{W_3^{\frac{1}{3}} + w_0}{D}\right)^2 \left(\frac{W_3}{W_3^{2/3}} + w_0\right) = \frac{\left(W_3^{\frac{1}{3}} + w_0\right)^3}{D^2},$$

which concludes the proof. ■

Corollary 5.2 (equivalent tasks for speed)**.** *Consider a fork or join graph with tasks T_i, $0 \leq i \leq n$, and a deadline D, and assume that the speeds in the optimal solution to* MINENERGY(G, D) *do not exceed f_{max}. Then, these speeds are the same as in the optimal solution for $n + 1$ independent tasks $T'_0, T'_1, \ldots, T'_n$, where $w'_0 = \left(\sum_{i=1}^{n} w_i^3\right)^{\frac{1}{3}} + w_0$, and, for $1 \leq i \leq n$, $w'_i = w'_0 \cdot \frac{w_i}{\left(\sum_{i=1}^{n} w_i^3\right)^{\frac{1}{3}}}$.*

Corollary 5.3 (equivalent task for energy)**.** *Consider a fork or join graph G and a deadline D, and assume that the speeds in the optimal solution to* MINENERGY(G, D) *do not exceed f_{max}.*

We say that the graph G is equivalent *to the graph $G^{(eq)}$, consisting of a single task $T_0^{(eq)}$ of weight $w_0^{(eq)} = \left(\sum_{i=1}^{n} w_i^3\right)^{\frac{1}{3}} + w_0$, because the minimum energy consumption of both graphs are identical:* $\mathbf{minE}(G, D) = \mathbf{minE}(G^{(eq)}, D)$.

Trees. We extend the results on a fork graph for a tree $G = (V, E)$ with $|V| = n + 1$ tasks. Let T_0 be the root of the tree; it has k children tasks, which are each themselves the root of a tree. A tree can therefore be seen as a fork graph, where the tasks of the fork are trees.

The previous results for fork graphs naturally lead to an algorithm that peels off branches of the tree, starting with the leaves, and replaces each fork subgraph in the tree, composed of a root T_0 and k children, by one task (as in Corollary 5.3) that becomes the unique child of T_0's parent in the tree. We say that this task is *equivalent* to the fork graph, since the optimal energy consumption will be the same. The computation of the *equivalent* weight of this task is done thanks to a call to the **eq** procedure, while the

tree procedure computes the solution to MINENERGY(G, D) (see Algorithm 2). Note that the algorithm computes the minimum energy for a tree, but it does not return the speeds at which each task must be executed. However, the algorithm returns the speed of the root task, and it is then straightforward to compute the speed of each children of the root task, and so on.

Theorem 5.2 (tree graphs)**.** *When G is a tree rooted in T_0 ($T_0 \in V$, where V is the set of tasks), the optimal solution to* MINENERGY(G, D) *can be computed in polynomial time $O(|V|^2)$.*

Algorithm 2: Solution to MINENERGY(G, D) for trees.

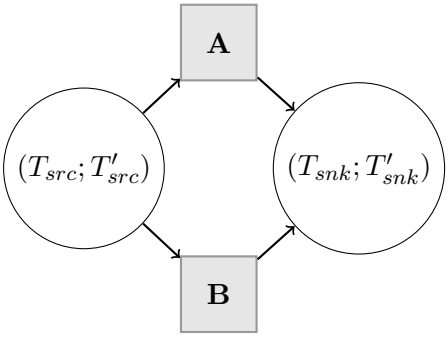
```
procedure tree (tree G, root T_0, deadline D)
begin
    Let w=eq (tree G, root T_0);
    if w/D ≤ f_max then
        return w^3/D^2;
    else
        if w_0/f_max > D then
            return Error:No Solution;
        else
            /* T_0 is executed at speed f_max */
            return w_0 × f_max^2 + Σ_{G_i subtree rooted in T_i∈children(T_0)} tree(G_i, T_i, D − w_0/f_max);

procedure eq (tree G, root T_0)
begin
    if children(T_0)=∅ then
        return w_0;
    else
        return ( Σ_{G_i subtree rooted in T_i∈children(T_0)} (eq(G_i, T_i))^3 )^{1/3} + w_0;
```

Proof. Let G be a tree graph rooted in T_0. The optimal solution to MINENERGY(G, D) is obtained with a call to **tree** (G, T_0, D), and we prove its optimality recursively on the depth of the tree. Similarly to the case of the fork graphs, we reduce the tree to an equivalent task that, if executed alone within a deadline D, consumes exactly the same amount of energy. The procedure **eq** is the procedure that reduces a tree to its equivalent task (see Algorithm 2).

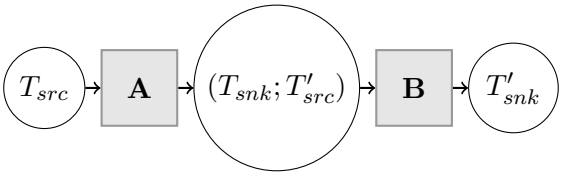
If the tree has depth 0, then it is a single task, **eq** (G, T_0) returns the equivalent weight w_0, and the optimal execution speed is $\frac{w_0}{D}$ (see Proposition 5.1). There is a solution if and only if this speed is not greater than $f_{\max}$, and then the corresponding energy consumption is $\frac{w_0^3}{D^2}$, as returned by the algorithm.

Assume now that for any tree of depth $i < p$, **eq** computes its equivalent weight, and **tree** returns its optimal energy consumption. We consider a tree G of depth p rooted in T_0: $G = T_0 \cup \{G_i\}$, where each subgraph G_i is a tree, rooted in T_i, of maximum depth $p - 1$. As in the case of forks, we know that each subtree G_i has a deadline $D - x$, where $x = \frac{w_0}{f_0}$, and f_0 is the speed at which task T_0 is executed. By induction hypothesis, we suppose that each graph G_i is equivalent to a single task, T_i', of weight w_i'

(a) Two SPGs before composition.



(b) Parallel composition.



(c) Series composition.

Figure 5.2: Composition of series-parallel graphs (SPGs).

(as computed by the procedure **eq**). We can then use the results obtained on forks to compute $w_0^{(eq)}$ (see proof of Theorem 5.1):

$$w_0^{(eq)} = \left(\sum_i (w'_i)^3 \right)^{\frac{1}{3}} + w_0.$$

Finally the tree is equivalent to one task of weight $w_0^{(eq)}$, and if $\frac{w_0^{(eq)}}{D} \leq f_{\max}$, the energy consumption is $\frac{\left(w_0^{(eq)}\right)^3}{D^2}$, and no speed exceeds $f_{\max}$.

Note that the speed of a task is always greater than the speed of its successors. Therefore, if $\frac{w_0^{(eq)}}{D} > f_{\max}$, we execute the root of the tree at speed $f_{\max}$ and then process each subtree G_i independently. Of course, there is no solution if $\frac{w_0}{f_{\max}} > D$, and otherwise we perform the recursive calls to **tree** to process each subtree independently. Their deadline is then $D - \frac{w_0}{f_{\max}}$.

To study the time complexity of this algorithm, first note that when calling **tree** (G, T_0, D), there might be at most $|V|$ recursive calls to **tree**, once at each node of the tree. Without accounting for the recursive calls, the **tree** procedure performs one call to the **eq** procedure, which computes the weight of the equivalent task. This **eq** procedure takes a time $O(|V|)$, since we have to consider the $|V|$ tasks, and we add the weights one by one. Therefore, the overall complexity is in $O(|V|^2)$. ■

Series-parallel graphs. We can further generalize our results to series-parallel graphs (SPGs), which are built from a sequence of compositions (parallel or series) of smaller-size SPGs. The smallest SPG consists of two nodes connected by an edge (such a graph is called an *elementary SPG*). The first node is the source, while the second one is the sink of the SPG. When composing two SGPs in series, we merge the sink of the first SPG with the source of the second one. For a parallel composition, the two sources are merged, as well as the two sinks, as illustrated in Figure 5.2.

We can extend the results for tree graphs to SPGs, by replacing step by step the SPGs by an equivalent task (procedure **weight** in Algorithm 3): we can compute the equivalent weight for a series or parallel composition.

However, since it is no longer true that the speed of a task is always larger than the speed of its successor (as was the case in a tree), we have not been able to find a recursive property on the tasks that should be set to $f_{\max}$, when one of the speeds obtained with the previous method exceeds $f_{\max}$. The problem of computing a closed form for a SPG with a finite value of $f_{\max}$ remains open. Still, we have the following result when $f_{\max} = +\infty$:

Theorem 5.3 (series-parallel graphs)**.** *When G is a SPG, it is possible to compute recursively a closed form expression of the optimal solution of* MINENERGY(G, D)*, assuming $f_{max} = +\infty$, in polynomial time $O(|V|)$, where V is the set of tasks.*

Algorithm 3: Solution to MINENERGY(G, D) for series-parallel graphs.

```
procedure SPG (series-parallel graph G, deadline D)
begin
    return (weight(G))^3 / D^2;

procedure weight (series-parallel graph G)
begin
    Let T_0 be the source of G and T_1 its sink;
    if G is composed of only two tasks, T_0 and T_1 then
        return w_0 + w_1;
    else
        /* G is a composition of two SPGs G_1 and G_2. */
        For i = 1, 2, let G'_i = G_i where the weight of source and sink tasks is set to 0;
        w'_1 = weight(G'_1); w'_2 = weight(G'_2);
        if G is a series composition then
            Let T_0 be the source of G_1, T_1 be its sink, and T_2 be the sink of G_2;
            return w_0 + w'_1 + w_1 + w'_2 + w_2;
        else
            /* It is a parallel composition. */
            Let T_0 be the source of G, and T_1 be its sink;
            return w_0 + ((w'_1)^3 + (w'_2)^3)^(1/3) + w_1;
```

Proof. Let G be a series-parallel graph. The optimal solution to MINENERGY(G, D) is obtained with a call to **SPG** (G, D), and we prove its optimality recursively. Similarly to trees, the main idea is to peel the graph off, and to transform it until there remains only a single equivalent task that, if executed alone within a deadline D, would consume exactly the same amount of energy. The procedure **weight** is the procedure that reduces a tree to its equivalent task (see Algorithm 3).

The proof is done by induction on the number of compositions required to build the graph G, p. If $p = 0$, G is an elementary SPG consisting in two tasks, the source T_0 and the sink T_1. It is therefore a linear chain, and therefore equivalent to a single task whose weight is the sum of both weights, $w_0 + w_1$ (see Corollary 5.1 for linear chains). The procedure **weight** returns therefore the correct equivalent weight, and **SPG** returns the minimum energy consumption.

Let us assume that the procedures return the correct equivalent weight and minimum energy consumption for any SPG consisting of $i < p$ compositions. We consider a SPG G, with p compositions. By definition, G is a composition of two smaller-size SPGs, G_1 and G_2, and both of these SPGs have

strictly fewer than p compositions. We consider G'_1 and G'_2, which are identical to G_1 and G_2, except that the weight of their source and sink tasks are set to 0 (these weights are handled separately), and we can reduce both of these SPGs to an equivalent task, of respective weights w'_1 and w'_2, by induction hypothesis. There are two cases:

- If G is a series composition, then after the reduction of G'_1 and G'_2, we have a linear chain in which we consider the source T_0 of G_1, the sink T_1 of G_1 (which is also the source of G_2), and the sink T_2 of G_2. The equivalent weight is therefore $w_0 + w'_1 + w_1 + w'_2 + w_2$, thanks to Corollary 5.1 for linear chains.
- If G is a parallel composition, the resulting graph is a fork-join graph, and we can use Corollaries 5.1 and 5.3 to compute the weight of the equivalent task, accounting for the source T_0 and the sink T_1: $w_0 + \left((w'_1)^3 + (w'_2)^3\right)^{\frac{1}{3}} + w_1$.

Once the weight of the equivalent task of the SPG has been computed with the call to **weight** (G), the optimal energy consumption is $\frac{(\mathbf{weight}(G))^3}{D^2}$.

Contrarily to the case of tree graphs, since we never need to call the **SPG** procedure again because there is no constraint on $f_{\max}$, the time complexity of the algorithm is the complexity of the **weight** procedure. There is exactly one call to **weight** for each composition, and the number of compositions in the SPG is in $O(|V|)$. All operations in **weight** can be done in $O(1)$, hence a complexity in $O(|V|)$. ■

General DAGs

For arbitrary execution graphs, we can rewrite the MINENERGY(G, D) problem as follows:

$$\begin{array}{lll} \text{Minimize} & & \sum_{i=1}^{n} u_i^{-2} \times w_i \\ \text{subject to} & \text{(i)} & b_i + w_i \times u_i \leq b_j \text{ for each edge } (T_i, T_j) \in \tilde{\mathcal{E}}, \\ & \text{(ii)} & b_i + w_i \times u_i \leq D \text{ for each task } T_i \in V, \\ & \text{(iii)} & u_i \geq \frac{1}{f_{\max}} \text{ for each task } T_i \in V, \\ & \text{(iv)} & b_i \geq 0 \text{ for each task } T_i \in V. \end{array} \tag{5.4}$$

Here, $u_i = 1/f_i$ is the inverse of the speed to execute task T_i. We now have a convex optimization problem to solve, with linear constraints in the non-negative variables u_i and b_i. In fact, the objective function is a posynomial, so we have a geometric programming problem [22, Section 4.5] for which efficient numerical schemes exist. In addition, such an optimization problem with a smooth convex objective function is known to be well-conditioned [92].

However, as illustrated on simple fork graphs, the optimal speeds are not expected to be rational numbers but instead arbitrarily complex expressions (we have the cubic root of the sum of cubes for forks, and nested expressions of this form for trees). From a computational complexity point of view, we do not know how to encode such numbers in polynomial size of the input (the rational task weights and the execution deadline). Still, we can always solve the problem numerically and get fixed-size numbers that are good approximations of the optimal values.

In the following, we show that the total power consumption of any optimal schedule is constant throughout execution. While this important property does not help to design an optimal solution, it shows that a schedule with large variations in its power consumption is likely to waste a lot of energy.

We need a few notations before stating the result. Consider a schedule for a graph $G = (V, E)$ with n tasks. Task T_i is executed at constant speed f_i (see Lemma 5.1) and during interval $[b_i, c_i]$: T_i begins

its execution at time b_i and completes it at time c_i. The total power consumption $P(t)$ of the schedule at time t is defined as the sum of the power consumed by all tasks executing at time t:

$$P(t) = \sum_{1 \leq i \leq n,\ t \in [b_i, c_i]} f_i^3 \ .$$

Theorem 5.4. *Consider an instance of* CONTINUOUS*, and an optimal schedule for this instance, such that no speed is equal to f_{max}. Then the total power consumption of the schedule throughout execution is constant.*

Proof. We prove this theorem by induction on the number of tasks of the graph. First we prove a preliminary result:

Lemma 5.2. *Consider a graph $G = (V, E)$ with $n \geq 2$ tasks, and any optimal schedule of deadline D. Let t_1 be the earliest completion time of a task in the schedule. Similarly, let t_2 be the latest starting time of a task in the schedule. Then, either G is composed of independent tasks, or $0 < t_1 \leq t_2 < D$.*

Proof. Task T_i is executed at speed f_i and during interval $[b_i, c_i]$. We have $t_1 = \min_{1 \leq i \leq n} c_i$ and $t_2 = \max_{1 \leq i \leq n} b_i$. Clearly, $0 \leq t_1, t_2 \leq D$ by definition of the schedule. Suppose that $t_2 < t_1$. Let T_1 be a task that ends at time t_1, and T_2 one that starts at time t_2. Then:

- $\nexists T \in V,\ (T_1, T) \in E$ (otherwise, T would start after t_2), therefore, $t_1 = D$;
- $\nexists T \in V,\ (T, T_2) \in E$ (otherwise, T would finish before t_1); therefore $t_2 = 0$.

This also means that all tasks start at time 0 and end at time D. Therefore, G is only composed of independent tasks. ■

Back to the proof of the theorem, we consider first the case of a graph with only one task. In an optimal schedule, the task is executed in time D, and at constant speed (Lemma 5.1), hence with constant power consumption.

Suppose now that the property is true for all DAGs with at most $n-1$ tasks. Let G be a DAG with n tasks. If G is exactly composed of n independent tasks, then we know that the power consumption of G is constant (because all task speeds are constant). Otherwise, let t_1 be the earliest completion time, and t_2 the latest starting time of a task in the optimal schedule. Thanks to Lemma 5.2, we have $0 < t_1 \leq t_2 < D$.

Suppose first that $t_1 = t_2 = t_0$. There are three kinds of tasks: those beginning at time 0 and ending at time t_0 (set S_1), those beginning at time t_0 and ending at time D (set S_2), and finally those beginning at time 0 and ending at time D (set S_3). Tasks in S_3 execute during the whole schedule duration, at constant speed, hence their contribution to the total power consumption $P(t)$ is the same at each time-step t. Therefore, we can suppress them from the schedule without loss of generality. Next we determine the value of t_0. Let $A_1 = \sum_{T_i \in S_1} w_i^3$, and $A_2 = \sum_{T_i \in S_2} w_i^3$. The energy consumption between 0 and t_0 is $\frac{A_1}{t_0^2}$, and between t_0 and D, it is $\frac{A_2}{(D-t_0)^2}$. The optimal energy consumption is obtained with $t_0 = \frac{A_1^{\frac{1}{3}}}{A_1^{\frac{1}{3}} + A_2^{\frac{1}{3}}}$. Then, the total power consumption of the optimal schedule is the same in both intervals, hence at each time-step: we derive that $P(t) = \left(\frac{A_1^{\frac{1}{3}} + A_2^{\frac{1}{3}}}{D} \right)^3$, which is constant.

Suppose now that $t_1 < t_2$. For each task T_i, let w_i' be the number of operations executed before t_1, and w_i'' the number of operations executed after t_1 (with $w_i' + w_i'' = w_i$). Let G' be the DAG G with execution weights w_i', and G'' be the DAG G with execution weights w_i''. The tasks with a weight equal to 0 are removed from the DAGs. Then, both G' and G'' have strictly fewer than n tasks. We

can therefore apply the induction hypothesis. We derive that the power consumption in both DAGs is constant. Since we did not change the speeds of the tasks, the total power consumption $P(t)$ in G is the same as in G' if $t < t_1$, hence a constant. Similarly, the total power consumption $P(t)$ in G is the same as in G'' if $t > t_1$, hence a constant. Considering the same partitioning with t_2 instead of t_1, we show that the total power consumption $P(t)$ is a constant before t_2, and also a constant after t_2. But $t_1 < t_2$, and the intervals $[0, t_2]$ and $[t_1, D]$ overlap. Altogether, the total power consumption is the same constant throughout $[0, D]$, which concludes the proof. ■

5.1.3 Conclusion

In this section, we have assessed the difficulty of energy-efficient schedules, in one of their simplest form: without reliability and with task preallocation. For the CONTINUOUS speeds model, we were able to conceive various algorithms for different special execution DAGs. Once again, we need to point out that the CONTINUOUS speed model is of conceptual importance, however this model cannot be achieved with physical devices. We have analyzed several more realistic speed models (with a finite number of possible speeds) in an extension of this work [J1].

Altogether, this section has laid theoretical foundations for a model taking energy into account. We now discuss how this energy efficiency impacts the reliability of an application.

5.2 Energy efficiency impacts reliability

Energy awareness is now recognized as a first-class constraint in the design of new scheduling algorithms. To help reduce energy dissipation, current processors from AMD, Intel and Transmeta allow the speed to be set dynamically, using a dynamic voltage and frequency scaling technique (DVFS). Indeed, a processor running at speed s dissipates s^3 watts per unit of time [6]. However, it has been recognized that reducing the speed of a processor has a negative effect on the reliability of a schedule: if a processor is slowed down, it has a higher chance to be subject to transient failures, caused for instance by software errors [134, 36].

In this part of the thesis, we investigate trade-offs between execution time and energy consumption for the execution of parallel applications on future systems. In Chapters 6, 7 and 8, we consider the DVFS technique introduced in Chapter 5.1. While energy consumption can be reduced by using speed scaling techniques, multiple authors [134, 36] showed that a reduced speed increases the number of transient fault rates of the system. In order to make up for the loss in reliability due to the energy efficiency, different models have been proposed for fault tolerance:

- *Re-execution [134, 99]:* this model consists in re-executing a task that does not meet the reliability constraint. This is the model under study in Chapters 6 and 7.
- *Replication [3, 53]:* this model consists in executing the same task on p different processors simultaneously, in order to meet the reliability constraints. This is also the model under study in Chapter 7, which combines re-execution and replication.
- *Checkpointing [85, 126]:* this model introduced in Part I, consists in "saving" the work done at some certain points of the work, hence reducing the amount of work lost when a failure occurs. This is the model under study in Chapter 8.

In Chapter 9, we consider a model closer to the reality of future Exascale platforms. The de-facto general-purpose error recovery technique in high performance computing is the coordinated checkpointing technique and rollback recovery, summarized in Part I, where all processes periodically stop computing and synchronize to write critical application data onto stable storage. We revisit in Chapter 9 the coordinated checkpointing mechanism, to obtain optimal checkpointing periods depending on the

fraction of power $\mathcal{P}_{\text{Cal}}$ spent when computing (by the CPUs), the fraction of power $\mathcal{P}_{\text{I/O}}$ spent when checkpointing and the base (or idle) power $\mathcal{P}_{\text{Static}}$, that is the power of having a processor turned on.

5.3 Related work

Reducing the energy consumption of computational platforms is an important research topic, and many techniques at the process, circuit design, and micro-architectural levels have been proposed [77, 74, 56]. The dynamic voltage and frequency scaling (DVFS) technique has been extensively studied, since it may lead to efficient energy/performance trade-offs [66, 49, 8, 32, 70, 122, 118]. Current microprocessors (for instance, from AMD [2] and Intel [76]) allow the speed to be set dynamically. Indeed, by lowering supply voltage, hence processor clock frequency, it is possible to achieve important reductions in power consumption, without necessarily increasing the execution time. We first discuss different optimization problems that arise in this context. Then we review energy models. Finally, we discuss studies that tackle the energy problem under reliability constraints.

DVFS and optimization problems

When dealing with energy consumption, the most usual optimization function consists in minimizing the energy consumption, while ensuring a deadline on the execution time (i.e., a real-time constraint), as discussed in the following papers.

Okuma et al. [93] demonstrate that voltage scaling is far more effective than the shutdown approach, which simply stops the power supply when the system is inactive. Their target processor employs just a few discretely variable voltages. De Langen and Juurlink [75] discuss leakage-aware scheduling heuristics that investigate both dynamic voltage scaling (DVS) and processor shutdown, since static power consumption due to leakage current is expected to increase significantly. Chen et al. [30] consider parallel sparse applications, and they show that when scheduling applications modeled by a directed acyclic graph with a well-identified critical path, it is possible to lower the voltage during non-critical execution of tasks, with no impact on the execution time. Similarly, Wang et al. [118] study the slack time for non-critical jobs, they extend their execution time and thus reduce the energy consumption without increasing the total execution time. Kim et al. [70] provide power-aware scheduling algorithms for bag-of-tasks applications with deadline constraints, based on dynamic voltage scaling. Their goal is to minimize power consumption as well as to meet the deadlines specified by application users.

For real-time embedded systems, slack reclamation techniques are used. Lee and Sakurai [77] show how to exploit slack time arising from workload variation, thanks to a software feedback control of supply voltage. Prathipati [100] discusses techniques to take advantage of run-time variations in the execution time of tasks; it determines the minimum voltage under which each task can be executed, while guaranteeing the deadlines of each task. Then, experiments are conducted on the Intel StrongArm SA-1100 processor, which has eleven different frequencies, and the Intel PXA250 XScale embedded processor with four frequencies. The goal of Xu et al. [121] is to schedule a set of independent tasks, given a worst case execution cycle (WCEC) for each task, and a global deadline, while accounting for time and energy penalties when the processor frequency is changing. The frequency of the processor can be lowered when some slack is obtained dynamically, typically when a task runs faster than its WCEC. Yang and Lin [122] discuss algorithms with preemption, using DVS techniques; substantial energy can be saved using these algorithms, which succeed to claim the static and dynamic slack time, with little overhead.

Since an increasing number of systems are powered by batteries, maximizing battery life also is an important optimization problem. Battery-efficient systems can be obtained with similar techniques

of dynamic voltage and frequency scaling, as described by Lahiri et al. [74]. Another optimization criterion is the energy-delay product, since it accounts for a trade-off between performance and energy consumption, as for instance discussed by Gonzalez and Horowitz [54]. We do not discuss further these latter optimization problems, since our goal is to minimize the energy consumption, with a fixed deadline.

Energy models

Several energy models are considered in the literature, and they can all be categorized in one of three models CONTINUOUS, DISCRETE, or VDD-HOPPING.

We quickly describe the two models that are not studied here:

DISCRETE **model.** Processors have a discrete number of predefined speeds (or frequencies), which correspond to different voltages that the processor can be subjected to [93]. Switching frequencies is not allowed during the execution of a given task, but two different tasks scheduled on a same processor can be executed at different frequencies.

VDD-HOPPING model. This model is similar to the DISCRETE one, except that switching modes during the execution of a given task is allowed: any rational speed can be simulated, by simply switching, at the appropriate time during the execution of a task, between two consecutive modes [87]. Note that V_{DD} usually represents the supply voltage, hence the name VDD-HOPPING.

The CONTINUOUS model is used mainly for theoretical studies. For instance, Yao et al. [123], followed by Bansal et al. [8], aim at scheduling a collection of tasks (with release time, deadline and amount of work), and the solution is the time at which each task is scheduled, but also, the speed at which the task is executed. In these papers, the speed can take any value, hence following the CONTINUOUS model.

We believe that the most widely used model is the DISCRETE one. Indeed, processors have currently only a few discrete number of possible frequencies [2, 76, 93, 100]. Therefore, most of the papers discussed above follow this model. Some studies exploit the continuous model to determine the smallest frequency required to run a task, and then choose the closest upper discrete value, as for instance [100] and [127].

Recently, a new local dynamic voltage scaling architecture has been developed, based on the VDD-HOPPING model [87, 9, 10]. Lee and Sakurai [77] showed that significant power can be saved by using two distinct voltages, and architectures using this principle have been developed (see for instance [68]). Compared to traditional power converters, a new design with no needs for large passives or costly technological options has been validated in a STMicroelectronics CMOS 65nm low-power technology [87].

Note that in the extension of this work [J1], we have compared these different models: on the one hand, we were able to assess the impact of the model on the problem complexity (polynomial vs NP-hard), and on the other hand, we were able to provide approximation algorithms building upon these results. The closest work to ours is the paper by Zhang et al. [127], in which the authors also consider the mapping of directed acyclic graphs, and compare the DISCRETE and the CONTINUOUS models. We go beyond their work in [J1], with an exhaustive complexity study, closed-form formulas for the continuous model and the comparison with the VDD-HOPPING models.

Reliability model

Since the introduction of DVFS, many papers have dealt with the optimization of energy consumption while enforcing a deadline [6, 8, 31] without reliability constraints.

Even though it has been pointed out that DVFS increases the probability of failures exponentially, and that this probability cannot be neglected in large-scale computing [96], very few authors have tackled the problem of optimizing the three criteria simultaneously: makespan, reliability and energy. The closest works to ours are [99, 134, 3].

Izosinov et al [99] study a tri-criteria optimization problem. They consider heterogeneous architectures. However, the DAG is already mapped on the architecture and each processor has the same set of speeds, so the fact that the architecture is heterogeneous does not really appear. They do not have any formal energy model. Furthermore, they assume that the user will specify the maximum number of failures per processor tolerated to satisfy the reliability constraint.

Zhu et al [134] are also addressing a tri-criteria optimization problem: minimizing the energy consumption while enforcing a deadline and matching reliability constraints. In order to do so, they choose some tasks that will have to be re-executed in order to match the reliability constraint. They simplify the scheduling problem by working on a single processor (as we will see, the problem is still NP-complete).

Finally, Assayad et al [3] have recently proposed an off-line tri-criteria scheduling heuristic (TSH). TSH uses active replication to minimize the schedule length, its global failure rate and its power consumption. They work on a homogeneous platform, fully connected. TSH is an improved critical-path list scheduling heuristic that takes into account power and reliability before deciding which task to assign and to duplicate onto the next free processors. The complexity of this heuristic is unfortunately exponential in the number of processors. Their heuristic however has the advantage of taking communication costs between processors into account.

Very few papers consider the general problem of the interplay between energy consumption and fault-tolerance on HPC platforms. There are three recent papers on this subject: Diouri et al. [37, 38] and Meneses et al. [86]. In their first paper [37], Diouri et al. presented the energy consumption of the three most important parts of fault-tolerance: message-logging, checkpointing and task coordination. Their first result it that task coordination is the most energy consuming part of fault-tolerance protocols. They also show that while it involves more power to store data on RAM, HDD logging is more energy consuming than RAM logging because of the logging duration. In their second paper, Diouri et al. [38] extend those results into a framework, ECOFIT, that predicts the energy consumption of a fault-tolerance protocol, allowing the user to choose amongst three fault-tolerance protocols: coordinated, uncoordinated and hierarchical depending on the application running on the platform. Meneses et al. [86] study the energy consumption of the coordinated periodic checkpointing protocol as a function of $\mathcal{P}_{\text{Static}}$ and $\mathcal{P}_{\text{Cal}}$.

Chapter 6

Energy-aware scheduling under reliability and makespan constraints

6.1 Introduction

Energy-aware scheduling has proven an important issue in the past decade, both for economical and environmental reasons. This holds true for traditional computer systems, not even to speak of battery-powered systems. More precisely, a processor running at speed s consumes s^3 watts per unit of time [6, 8, 31], hence it consumes $s^3 \times d$ joules when operated during d units of time. To help reduce energy dissipation, processors can run at different speeds. A widely used technique to reduce energy consumption is *dynamic voltage and frequency scaling (DVFS)*, also known as speed-scaling [6, 8, 31]. We consider here the popular CONTINUOUS model for speeds, but in an extension of this work [C2] we have considered models with a discrete set of speeds.

While energy consumption can be reduced by using speed scaling techniques, it was shown [134, 36] that speed scaling increases the number of transient fault rates of the system. In order to make up for the loss in reliability due to the energy, we propose to use in this chapter the *re-execution* model [134, 99] for fault tolerance.

Consider a Directed Acyclic Graph (DAG) of n tasks that has to be computed on p identical processors. The traditional scheduling objective consists in minimizing the execution time, or makespan, to process the DAG. In order to do so, the DAG is mapped on the processors and an execution speed is assigned to each task of the DAG. Using these speeds, we can define an energy and a reliability for the DAG.

Main contributions. In this chapter, we present theoretical results to better understand the tri-criteria optimization problem. We show the intractability of the problem on the simplest graph: a linear chain, but show that there are graphs (fork graphs) where the problem can be solved in polynomial time. We show that the optimal strategy for linear chains and fork graphs are very different, and based on this, we present novel tri-criteria heuristics that use re-execution in order to minimize the energy consumption under the constraints of both a reliability threshold and a deadline bound.

The rest of the chapter is organized as follows. The first contribution is a formal model of the tri-criteria scheduling problem under consideration (Section 6.2). The second contribution is to provide theoretical results for the CONTINUOUS speed models (Section 6.3). The third contribution is the design of tri-criteria scheduling heuristics that use re-execution to increase the reliability of a system (Section 6.4),

and their evaluation through extensive simulations (Section 6.5). To the best of our knowledge, this is the first attempt to propose practical solutions to this tri-criteria problem. Finally, we give concluding remarks in Section 6.6.

6.2 Model

The model in this chapter is an extension of the model presented in Section 5.1.1. Consider an application task graph $\mathcal{G} = (V, \mathcal{E})$, where the vertex set $V = \{T_1, \ldots, T_n\}$ is the set of n tasks, and the edge set $\mathcal{E}$ corresponds to the precedence constraints between tasks. For $1 \leq i \leq n$, task T_i has a weight w_i, that corresponds to the computation requirement of the task. We assume that we have a parallel platform made up of p identical processors. Each processor has a set of available speeds ($[f_{\min}, f_{\max}]$). The goal is to the minimize the energy consumed during the execution of the graph while enforcing a deadline bound and matching a reliability threshold. To match the reliability threshold, some tasks will be executed once, and some tasks will be re-executed. We detail below the conditions that are enforced on the corresponding execution speeds.

In this section, for the sake of clarity, we assume that a task is executed at the same (unique) speed throughout execution. In Section 6.3, we show that this strategy is optimal.

6.2.1 Makespan

The makespan of a schedule is its total *execution time*. The first task is scheduled at time 0, so that the makespan of a schedule is simply the maximum time at which one of the processors finishes its computations. We consider a deadline bound D, which is a constraint on the makespan.

The main difference with Section 5.1.1 is that in this chapter, a schedule specifies the tasks that are re-executed, and the speed at which each task is executed (and possibly re-executed), and its makespan should not be greater than D.

When a task is scheduled to be re-executed at two different speeds f_i^1 and f_i^2, we always account for both executions, even when the first execution is successful: in other words, we consider a worst-case execution scenario, and the deadline D must be matched even in the case where all tasks that are re-executed fail during their first execution. The execution time of a task T_i of weight w_i at speed f_i is $\mathcal{E}xe(w_i, f_i) = \frac{w_i}{f_i}$.

6.2.2 Reliability

We use the fault model of Zhu and Aydin [133]. *Transient* failures are faults caused by soft errors for example. They invalidate only the execution of the current task and the processor subject to that failure will be able to recover and execute the subsequent task assigned to it (if any). In addition, we use the reliability model introduced by Shatz and Wang [113], which states that the radiation-induced transient faults follow a Poisson distribution. The parameter λ of the Poisson distribution is then:

$$\lambda(f) = \tilde{\lambda_0}\, e^{\tilde{d}\frac{f_{\max} - f}{f_{\max} - f_{\min}}}, \tag{6.1}$$

where $f_{\min} \leq f \leq f_{\max}$ is the processing speed, the exponent $\tilde{d} \geq 0$ is a constant, indicating the sensitivity of fault rates to DVFS, and $\tilde{\lambda_0}$ is the average fault rate corresponding to $f_{\max}$. We see that reducing the speed for energy saving increases the fault rate exponentially. The reliability of a task T_i executed once at speed f_i is:

$$R_i(f_i) = e^{-\lambda(f_i) \times \mathcal{E}xe(w_i, f_i)}.$$

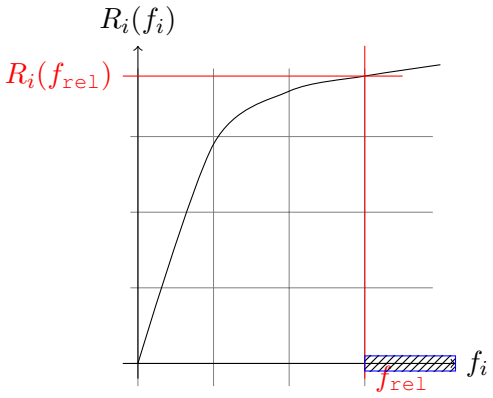


Figure 6.1: Constraint on reliability.

Because the fault rate is usually very small, of the order of 10^{-6} per time unit (Baleani et al. [7] or Izosinov et al. [99]), 10^{-5} (Assayad et al. [3]), we can use the first order approximation of $R_i(f_i)$ as

$$\begin{aligned} R_i(f_i) &= 1 - \lambda(f_i) \times \mathcal{E}xe(w_i, f_i) \\ &= 1 - \tilde{\lambda_0}\ e^{\tilde{d}\frac{f_{\max} - f_i}{f_{\max} - f_{\min}}} \times \frac{w_i}{f_i} \\ &= 1 - \lambda_0\ e^{-df_i} \times \frac{w_i}{f_i}, \end{aligned} \tag{6.2}$$

where $d = \frac{\tilde{d}}{f_{\max} - f_{\min}}$ and $\lambda_0 = \tilde{\lambda_0} e^{df_{\max}}$.

We want the reliability of each task T_i to be greater than a given threshold, namely $R_i(f_{\texttt{rel}})$ (see Figure 6.1).[1] If task T_i is executed only once, then $f_{\texttt{rel}}$ is the minimum speed at which T_i must be executed to match the reliability constraint (recall that the reliability of a task increases with its speed). If task T_i is re-executed, then let $f_i^{(1)}$ be the speed of the first execution and $f_i^{(2)}$ be the speed of the second execution. The execution of T_i is successful if and only if both attempts do not fail, so that the reliability of T_i with re-execution is $R_i = 1 - (1 - R_i(f_i^{(1)}))(1 - R_i(f_i^{(2)}))$, and we want this quantity

[1] It is possible to compute $f_{\texttt{rel}}$, given a system reliability requirement R_0: the reliability of a system (the probability that all tasks are executed successfully) where all tasks are executed once at speed $f_{\texttt{rel}}$ is

$$\begin{aligned} R_{\text{sys}} &= \Pi_{i=1}^{n} R_i(f_{\texttt{rel}}) \\ &= e^{-\lambda(f_{\texttt{rel}}) \sum_{i=1}^{n} \mathcal{E}xe(w_i, f_{\texttt{rel}})} \\ &= e^{-\frac{\lambda(f_{\texttt{rel}})}{f_{\texttt{rel}}} \sum_{i=1}^{n} w_i} \end{aligned}$$

Then $R_{\text{sys}} \geq R_0$:

$$\begin{aligned} e^{-\frac{\lambda(f_{\texttt{rel}})}{f_{\texttt{rel}}} \sum_{i=1}^{n} w_i} &\geq R_0 \\ \frac{e^{-df_{\texttt{rel}}}}{f_{\texttt{rel}}} &\leq \frac{-\ln R_0}{\lambda_0 \sum_{i=1}^{n} w_i} \\ f_{\texttt{rel}} &\geq \frac{W\left(\frac{\lambda_0 d \sum_{i=1}^{n} w_i}{-\ln R_0}\right)}{d} \end{aligned}$$

where W is the product logarithmic (Lambert) function.

to be at least equal to $R_i(f_{\texttt{rel}})$. If task T_i is executed only once at speed f_i, we let $R_i = R_i(f_i)$, and the reliability constraint finally writes:

$$\text{RELIABILITY}: \quad R_i \geq R_i(f_{\texttt{rel}}) \text{ for } 1 \leq i \leq n \ . \tag{6.3}$$

6.2.3 Energy

If task T_i is executed once at speed f_i, we have seen that the consumed energy is

$$E_i = E_i(f_i) = \mathcal{E}xe(w_i, f_i) \times f_i^3. \tag{6.4}$$

If task T_i is re-executed at speeds $f_i^{(1)}$ and $f_i^{(2)}$, it is natural to add up the energy consumed during both executions, just as we add up both execution times when enforcing the makespan deadline. Again, this corresponds to the worst-case execution scenario. We obtain:

$$E_i = E_i(f_i^{(1)}) + E_i(f_i^{(2)}). \tag{6.5}$$

Note that some authors [133] consider only the energy spent for the first execution, which seems unfair: re-execution comes at a price both in the deadline and in the energy consumption. The total energy consumed by the schedule is

$$\text{ENERGY}: \quad E = \sum_{i=1}^{n} E_i \ , \tag{6.6}$$

where E_i is defined by Equation (6.4) if T_i is not re-executed, and by Equation (6.5) otherwise.

6.2.4 Optimization problem

Definition 6.1. TRI-CRIT-CONT. Given an application graph $\mathcal{G} = (V, \mathcal{E})$ and p homogeneous processors with continuous speeds, TRI-CRIT-CONT is the problem of minimizing the energy consumption in Equation (6.6), subject to the deadline bound D and to the reliability constraint of Equation (6.3).

6.3 CONTINUOUS model

If we assume that $f_{\text{min}} = f_{\texttt{rel}} = f_{\text{max}}$, we can easily reduce TRI-CRIT-CONT to a classical scheduling problem that is NP-complete as soon as there are two processors [48]. However this result is not very interesting. We want to show that even without considering the mapping issue, the problem is still NP-hard. In order to do so, in this section we consider *linear chains of tasks*. For a linear chain, we have $\mathcal{E} = \cup_{i=1}^{n-1}\{T_i \rightarrow T_{i+1}\}$, and any list scheduling mapping is always optimal, for any processor number (there is always only one order to map the tasks).

In this section, after establishing the optimality of unique-speed execution per task, we show that finding the solution of TRI-CRIT-CONT on a linear chain and with a single processor is NP-hard. We conclude with additional results that will guide the design of the heuristics in Section 6.4.

6.3.1 Optimality of unique-speed execution per task

Lemma 6.1. *With the* TRI-CRIT-CONT *model, it is optimal to execute each task at a unique speed throughout its execution.*

Proof. First let us assume that the function that gives the speed of the execution of a task is a piecewise-constant function. The general proof is a direct corollary from the theorem that states that any piecewise-continuous function defined on an interval $[a, b]$ can be uniformly approximated as closely as desired by a piecewise-constant function [106]. Therefore, this proof is valid for any piecewise-continuous function.

Suppose that in the optimal solution, there is a task whose speed changes during the execution. Consider the first time-step at which the change occurs: the computation begins at speed f from time t to time t', and then continues at speed f' until time t''. The total energy consumption for this task in the time interval $[t, t'']$ is $E = (t' - t) \times f^3 + (t'' - t') \times (f')^3$. Moreover, the amount of work done for this task is $W = (t' - t) \times f + (t'' - t') \times f'$. The reliability of the task is exactly $1 - \lambda_0 \left((t' - t) \times e^{-df} + (t'' - t') \times e^{-df'} + r \right)$, where r is a constant due to the reliability of the rest of the process, which is independent from what happens during $[t, t'']$. The reliability is a function that increases when the function $h(t, t', t'', f, f') = (t' - t) \times e^{-df} + (t'' - t') \times e^{-df'}$ decreases.

If we run the task during the whole interval $[t, t'']$ at constant speed $W/(t'' - t)$, the same amount of work is done within the same time, and the energy consumption during this interval of time becomes $E' = (t'' - t) \times (W/(t'' - t))^3$. Note that the new speed can be expressed as $f_d = af + (1 - a)f'$, where $0 < a = \frac{t'-t}{t''-t} < 1$. Therefore, because of the convexity of the function $x \mapsto x^3$, we have $E' < E$. Similarly, since $x \mapsto e^{-dx}$ is a convex function, $h(t, t', t'', f, f') < h(t, t', t'', f_d, f_d)$, and the reliability constraint is also matched. This contradicts the hypothesis of optimality of the first solution, and concludes the proof. ■

Next we show that not only a task is executed at a single speed, but that its re-execution (when it occurs) is executed at the same speed as the first execution:

Lemma 6.2. *With the* TRI-CRIT-CONT *model, it is optimal to re-execute each task (whenever needed) at the same speed as its first execution.*

Proof. Consider a task T_i executed a first time at speed f_i, and a second time at speed $f_i' > f_i$. Assume first that $d = 0$, i.e., the reliability of task T_i executed at speed f_i is $R_i(f_i) = 1 - \lambda_0 \frac{w_i}{f_i}$. We show that executing task T_i twice at speed $f = \sqrt{f_i f_i'}$ improves the energy consumption while matching the deadline and reliability constraints. Clearly the reliability constraint is matched, since $1 - \lambda_0^2 w_i^2 \frac{1}{f^2} = 1 - \lambda_0^2 w_i^2 \frac{1}{f_i f_i'}$. The fact that the deadline constraint is matched is due to the fact that $\sqrt{f_i f_i'} \geq \frac{2 f_i f_i'}{f_i + f_i'}$ (by squaring both sides of the equation we obtain $(f_i - f_i')^2 \geq 0$). Then we use the fact that $f_d = \frac{2 f_i f_i'}{f_i + f_i'}$ is the minimal speed such that $\forall f \geq f_d$, $\frac{2 w_i}{f} < \frac{w_i}{f_i} + \frac{w_i}{f_i'}$. Finally, it is easy to see that the energy consumption is improved since $2 f_i f_i' \leq f_i^2 + f_i'^2$, hence $2 w_i f_i f_i' \leq w_i f_i^2 + w_i f_i'^2$.

In the general case when $d \neq 0$, instead of having a closed form formula for the new speed f common to both executions, we have $f = \max(f_1, f_2)$, where f_1 is dictated by the reliability constraint, while f_2 is dictated by the deadline constraint. f_1 is the solution to the equation $2(dX + \ln X) = (df_i + \ln f_i) + (df_i' + \ln f_i')$; this equation comes from the reliability constraint: the minimum speed X to match the reliability is obtained with $1 - \lambda_0^2 w_i^2 \frac{e^{-df_i}}{f_i} \frac{e^{-df_i'}}{f_i'} = 1 - \lambda_0^2 w_i^2 \frac{e^{-2dX}}{X^2}$. The deadline constraint must also be enforced, and hence $f_2 = \frac{2 f_i f_i'}{f_i + f_i'}$ (minimum speed to match the deadline). Then the fact that the energy does not increase comes from the convexity of this function. ■

Note that the unique-speed result applies to any solution of the problem, not just optimal solutions, hence all heuristics of Section 6.4 will assign a unique speed to each task, be it re-executed or not.

6.3.2 Intractability of TRI-CRIT-CONT

Theorem 6.1. *The* TRI-CRIT-CONT *problem is NP-hard, but not known to be in NP.*

Proof. Consider the associated decision problem: given a deadline, and energy and reliability bounds, can we schedule the graph to match all these bounds? Since the speeds could take any real values, the problem is not known to be in NP. For the completeness, we use a reduction from SUBSET-SUM [48]. Let $\mathcal{I}_1$ be an instance of SUBSET-SUM: given n strictly positive integers $a_1, \ldots, a_n$, and a positive integer X, does there exist a subset I of $\{1, \ldots, n\}$ such that $\sum_{i \in I} a_i = X$? Let $S = \sum_{i=1}^{n} a_i$.

We build the following instance $\mathcal{I}_2$ of our problem. The execution graph is a linear chain with n tasks, where:

- task T_i has weight $w_i = a_i$;
- $\lambda_0 = \frac{f_{\max}}{100 \max_i a_i}$;
- $f_{\min} = \sqrt{\lambda_0 \max_i a_i f_{\max}} = \frac{1}{10} f_{\max}$;
- $D_0 = \frac{S}{f_{\max}} + \frac{X}{c f_{\max}}$, where
  $c = 4\sqrt{\frac{2}{7}} \cos \frac{1}{3}(\pi - \tan^{-1} \frac{1}{\sqrt{7}}) - 1 \ (\approx 0.2838)$;
- $f_{\texttt{rel}} = f_{\max}$; $d = 0$; $R_i^0 = R_i^{f_{\texttt{rel}}} = 1 - \lambda_0 \frac{w_i}{f_{\texttt{rel}}}$;
- $E_0 = 2X(\frac{2c}{1+c} f_{\texttt{rel}})^2 + (S - X) f_{\texttt{rel}}^2$.

Clearly, the size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$.

Suppose first that instance $\mathcal{I}_1$ has a solution, I. For all $i \in I$, T_i is executed twice at speed $\frac{2c}{1+c} f_{\texttt{rel}}$. Otherwise, for all $i \notin I$, it is executed at speed $f_{\texttt{rel}}$ one time only. The execution time is $\sum_{i \notin I} \frac{a_i}{f_{\texttt{rel}}} + \sum_{i \in I} 2\frac{a_i}{\frac{2c}{1+c} f_{\texttt{rel}}} = \frac{S-X}{f_{\texttt{rel}}} + 2X\frac{1+c}{2c f_{\texttt{rel}}} = D_0$. The reliability constraint is obviously met for tasks not in I. It is also met for all tasks in I, since $\frac{2c}{1+c} f_{\texttt{rel}} > f_{\min}$ and two executions at $f_{\min}$ suffice to match the reliability constraint. Indeed, $1 - \lambda_0^2 \frac{a_i^2}{f_{\min}^2} = 1 - \lambda_0 \frac{a_i}{f_{\texttt{rel}}} \cdot \frac{a_i^2}{\max_i a_i} \geq 1 - \lambda_0 \frac{a_i}{f_{\texttt{rel}}} = R_i^0$. The energy consumption is exactly E_0. All bounds are respected, and therefore the execution speeds are a solution to $\mathcal{I}_2$.

Suppose now that $\mathcal{I}_2$ has a solution. Let $I = \{i \mid T_i \text{ is executed twice in the solution } \}$. Let $Y = \sum_{i \in I} a_i$. We prove in the following that necessarily $Y = X$ in order to match the energy constraint E_0.

We first point out that tasks executed only once are necessarily executed at maximum speed to match the reliability constraint. Then consider the problem of minimizing the energy of a set of tasks, some executed twice, some executed once at maximum speed, and assume that we have a deadline D_0 to match, but no constraint on reliability or on $f_{\min}$. We will verify later that these additional two constraints are indeed satisfied by the optimal solution when the only constraint is the deadline. Thanks to Corollary 6.2, for all $i \in I$, $f_i = f_i'$ (where f' is the speed of the second execution). Because the deadline is the only constraint, either $Y = 0$ (no tasks are re-executed), or it is optimal to match the deadline (otherwise we could just slow down one of the re-executed tasks and this would decrease the total energy). Hence the problem amounts to finding the values of Y and f that minimize the function $E = 2Y f^2 + (S - Y) f_{\texttt{rel}}^2$ with the constraint $(S - Y)/f_{\texttt{rel}} + 2Y/f \leq D_0$. First, note that if $Y = 0$ then $E > E_0$. Hence, we can assume that $Y > 0$. Then we have $f = \frac{2Y}{D_0 f_{\texttt{rel}} - (S-Y)} f_{\texttt{rel}}$ because of the previous remark (tight deadline). Plugging back the value of f in E, we have:

$$E(Y) = \left(\frac{(2Y)^3}{(D_0 f_{\texttt{rel}} - (S - Y))^2} + (S - Y) \right) f_{\texttt{rel}}^2,$$

which admits a unique minimum when $Y = c(D_0 f_{\texttt{rel}} - S) = X$. To see this, define $\tilde{Y} = \frac{Y}{D_0 f_{\texttt{rel}} - S}$. Then we have: $E(\tilde{Y}) = \left(\frac{(2\tilde{Y})^3}{(1+\tilde{Y})^2} + (\frac{S}{D_0 f_{\texttt{rel}} - S} - \tilde{Y}) \right) (D_0 f_{\texttt{rel}} - S) f_{\texttt{rel}}^2$. Differentiating, we obtain

$$E'(\tilde{Y}) = \left(\frac{3 \cdot 2^3 \tilde{Y}^2}{(1+\tilde{Y})^2} - \frac{2^4 \tilde{Y}^3}{(1+\tilde{Y})^3} - 1 \right) (D_0 f_{\texttt{rel}} - S) f_{\texttt{rel}}^2.$$

Then $E' = 0$ iff

$$24\tilde{Y}^2(1+\tilde{Y}) - 16\tilde{Y}^3 - (1+\tilde{Y})^3 = 0. \tag{6.7}$$

The only positive solution of Equation (6.7) is $\tilde{Y} = c$, hence the result. Note that when $Y = X$, then $E = E_0$, so any other value of Y will not be valid. There remains to check that the solution matches both constraints on $f_{\min}$ and on reliability. We have $f = \frac{2c}{1+c} f_{\texttt{rel}} > f_{\min}$, which shows that both constraints hold for task in I. Other tasks are executed at speed $f_{\texttt{rel}}$. Altogether, we have $\sum_{i \in I} a_i = Y = X$, and therefore $\mathcal{I}_1$ has a solution. This concludes the proof. ■

6.3.3 Additional results

Proposition 6.1. *Consider a linear chain execution on a single processor, with the* TRI-CRIT-CONT *model. Suppose $f_{\texttt{rel}} < f_{max}$. In any optimal solution, either all tasks are executed only once, and at constant speed $\max(\frac{\sum_{i=1}^n w_i}{D}, f_{\texttt{rel}})$, or at least one task is re-executed, and then all those tasks that are not re-executed are executed at speed $f_{\texttt{rel}}$.*

Proof. Consider an optimal schedule. If all tasks are executed only once, the smallest energy consumption is obtained when using the constant speed $\frac{\sum_{i=1}^n w_i}{D}$. However if $\frac{\sum_{i=1}^n w_i}{D} < f_{\texttt{rel}}$, then we have to execute all tasks at speed $f_{\texttt{rel}}$ to match both reliability and deadline constraints.

Now, assume that some task T_i is re-executed, and assume by contradiction, that some other task T_j is executed only once at speed $f_j > f_{\texttt{rel}}$. Note that the common speed f_i used in both executions of T_i is smaller than $f_{\texttt{rel}}$, otherwise we would not need to re-execute T_i. We have $f_i < f_{\texttt{rel}} < f_j$, and we prove that there exist values f_i' (new speed of T_i) and f_j' (new speed of T_j) such that $f_i < f_i' \leq f_{\texttt{rel}} \leq f_j' < f_j$, and the energy consumed with the new speeds is strictly smaller, while the execution time is unchanged. The constraint on reliability will also be met, since the speed of T_i is increased while the speed of T_j remains above the reliability threshold.

Our problem writes: does there exist $\varepsilon,\ \varepsilon' > 0$ such that:

$$\begin{aligned} w_i f_i^2 + w_j f_j^2 &> w_i (f_i + \varepsilon')^2 + w_j (f_j - \varepsilon)^2; \\ D = \frac{w_i}{f_i} + \frac{w_j}{f_j} &= \frac{w_i}{f_i + \varepsilon'} + \frac{w_j}{f_j - \varepsilon} \\ f_i < f_i + \varepsilon' &\leq f_{\texttt{rel}} \leq f_j - \varepsilon < f_j. \end{aligned}$$

This is equivalent to:

$$\begin{aligned} w_i f_i^2 + w_j f_j^2 &> w_i (f_i + \varepsilon')^2 + w_j (f_j - \varepsilon)^2 \\ \varepsilon' &= \frac{w_i}{D - \frac{w_j}{f_j - \varepsilon}} - f_i \\ 0 < \varepsilon &\leq f_j - \max\left(f_{\texttt{rel}}, \frac{w_i + w_j}{\frac{w_i}{f_i} + \frac{w_j}{f_j}} \right). \end{aligned}$$

Let $\phi : \varepsilon \mapsto w_i f_i^2 + w_j f_j^2 - \big(w_i(f_i+\varepsilon')^2 + w_j(f_j-\varepsilon)^2\big)$. Then $\phi(\varepsilon) = \frac{w_i^3 f_j^2}{(Df_j - w_j)^2} - \frac{w_i^3 (f_j-\varepsilon)^2}{(D(f_j-\varepsilon)-w_j)^2} + w_j f_j^2 - w_j(f_j-\varepsilon)^2$. We want to prove that ϕ is positive when $0 < \varepsilon \leq f_j - \max\left(f_{\texttt{rel}}, \frac{w_i+w_j}{\frac{w_i}{f_i}+\frac{w_j}{f_j}}\right)$. Differentiating, we obtain $\phi'(\varepsilon) =$

$$\frac{2w_i^3(f_j-\varepsilon)}{(D(f_j-\varepsilon)-w_j)^2} - \frac{2Dw_i^3(f_j-\varepsilon)^2}{(D(f_j-\varepsilon)-w_j)^3} + 2w_j(f_j-\varepsilon),$$

which simplifies into the polynomial

$$X^3 + 3X^2 + 3X + \frac{w_i^3}{w_j^3} + 1 = 0,$$

by multiplying each side of the equation by $\frac{(D(f_j-\varepsilon)-w_j)^3}{w_j^3(f_j-\varepsilon)}$, and defining $X = \frac{D(f_j-\varepsilon)-w_j}{w_j}$. The only real solution to this polynomial is $-\frac{w_i}{w_j} - 1 < 0$, hence $\forall \varepsilon > 0,\ \phi'(\varepsilon) < 0$. We deduce that ϕ is minimal when ε is maximal, that is: $\varepsilon = f_j - \max\left(f_{\texttt{rel}}, \frac{w_i+w_j}{\frac{w_i}{f_i}+\frac{w_j}{f_j}}\right)$.

Altogether we have found f_i' and f_j' that improve the energy consumption of the schedule. However it was supposed to be optimal, we have a contradiction. ■

In essence, Proposition 6.1 states that when dealing with a linear chain, we should first slow down the execution of each task as much as possible. Then, if the deadline is not too tight, i.e., if $f_{\texttt{rel}} > \frac{\sum_{i=1}^n w_i}{D}$, there remains the possibility to re-execute some of the tasks (and of course it is NP-hard to decide which ones). Still, this general principle *"first slow-down and then re-execute"* will guide the design of type A heuristics in Section 6.4.

Lemma 6.3. *With the* TRI-CRIT-CONT *model, when all the tasks of a DAG have the same weight w, then in any optimal solution, each task is executed (or re-executed) at least at speed f, with:*

$$\lambda_0 w \frac{e^{-2df}}{f} = \frac{e^{-df_{rel}}}{f_{rel}}. \tag{6.8}$$

Proof. Let us assume first that a task is executed only once at a speed $f_1 < f$. Then, the reliability constraint is not satisfied, and indeed we must have in this case $f_1 \geq f_{\texttt{rel}} > f$.

If the task is re-executed, with one execution at speed f_1 and the other at speed f_2, then if both f_1 and f_2 are strictly smaller than f, the reliability constraint is not satisfied too because of Equation (6.8). The last case to consider is such that $f_1 < f \leq f_2$, but in this case the solution would not be optimal since it is strictly better to execute the task twice at the same speed, following Lemma 6.2. ■

In the following, we consider a DAG made of identical tasks, and since a task will never be executed at a speed lower than the speed f defined by Equation (6.8), we assume that f_{min} is at least equal to f; otherwise, we can let $f_{\text{min}} = f$, thanks to Lemma 6.3.

Proposition 6.2. *Consider a fork graph of $n+1$ identical tasks (a source and $n \geq 2$ independent successors, all of same weight w), executed on $n+1$ processors, with the* TRI-CRIT-CONT *model. In the optimal solution, the source is executed at speed f_{src}, and the n successors are executed at the same speed f_{leaf}. If $D < \frac{2w}{f_{max}}$, then there is no solution. Otherwise, the number of executions and the values of f_{src} and f_{leaf} depend upon the deadline D as follows:*

1. **No task is re-executed:**
   - *if* $\frac{2w}{f_{max}} \leq D \leq \frac{w}{f_{max}}(1+n^{\frac{1}{3}})$, *then* $f_{src} = f_{max}$ *and* $f_{leaf} = \frac{w}{Df_{max}-w} f_{max}$;
   - *if* $\frac{w}{f_{max}}(1+n^{\frac{1}{3}}) < D \leq \frac{w}{f_{rel}} \frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$, *then* $f_{src} = \frac{w}{D}(1+n^{\frac{1}{3}})$ *and* $f_{leaf} = \frac{w}{D} \frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$;
   - *if* $\frac{w}{f_{rel}} \frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}} < D \leq \frac{2w}{f_{rel}}$, *then* $f_{src} = \frac{w}{Df_{rel}-w} f_{rel}$ *and* $f_{leaf} = f_{rel}$;
   - *if* $\frac{2w}{f_{rel}} < D \leq \frac{w}{f_{rel}} \frac{(1+2n^{\frac{1}{3}})^{\frac{3}{2}}}{\sqrt{1+n}}$, *then* $f_{src} = f_{leaf} = f_{rel}$.
2. **The source is executed once, and the successors are re-executed:**
   - *if* $\frac{w}{f_{rel}} \frac{(1+2n^{\frac{1}{3}})^{\frac{3}{2}}}{\sqrt{1+n}} < D \leq \frac{w}{f_{rel}}(1+2n^{\frac{1}{3}})$, *then* $f_{src} = \frac{w}{D}(1+2n^{\frac{1}{3}})$ *and* $f_{leaf} = \frac{w}{D} \frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$;
   - *if* $\frac{w}{f_{rel}}(1+2n^{\frac{1}{3}}) < D \leq \frac{w}{f_{rel}} 2\sqrt{2}(1+n^{\frac{1}{3}})$, *then* $f_{src} = f_{rel}$ *and* $f_{leaf} = \max(\frac{2w}{Df_{rel}-w} f_{rel}, f_{min})$.
3. **Each task is re-executed:** *this case is more difficult to characterize. Indeed, it depends on* f_{min}*, which is dependent on the system and on the weight* w *of the tasks (see Lemma 6.3). Hence, even when* $D > \frac{w}{f_{rel}} 2\sqrt{2}(1+n^{\frac{1}{3}})$*, the source is not necessarily re-executed, since it may be better to slow down all the successors, if it is possible to run them slow enough. We do not detail these cases.*

Proof. First we recall preliminary results that are valid for any optimal solution of the TRI-CRIT-CONT problem:

- the re-execution of any task will always be at the same speed as its execution;
- if a task is executed only once, then $f_{\mathtt{rel}} \leq f \leq f_{\max}$;
- if a task is re-executed, then $f_{\min} \leq f < \frac{1}{\sqrt{2}} f_{\mathtt{rel}}$.

Thanks to these preliminary results, we know that if two tasks of same weight w have the same energy consumption in the optimal solution, then they are executed the same number of times (once or twice) and at the same speed: when the number of execution is the same, the bijection Energy-Speed is obvious. Because of the intervals of speed depending on the number of execution, we see that for a given energy, if the energy is greater or equal than $wf_{\mathtt{rel}}^2$ then necessarily there is one execution, if it is lower than $wf_{\mathtt{rel}}^2$ then necessarily there are two executions.

We prove that in any solution, the energy consumed for the execution of each successor task, also called *leaf*, is the same. If it was not the case, since each task has the same weight, and since each leaf is independent from the other and dependent on the source of the fork, if a leaf T_i is consuming more than another leaf T_j, then we could execute T_i the same number of times and at the same speed than T_j, hence matching the deadline bound and the reliability constraint, and obtaining a better solution. Thanks to this result, we now assume that all leaves are executed at the same speed(s), denoted $f_{\mathtt{leaf}}$. The source task may be executed at a different speed, $f_{\mathtt{src}}$.

Next, let us show that the energy consumption of the source is always greater or equal than the one from any leaf in any optimal solution. First, since the source and leaves have the same weight, if we invert the execution speed(s) of the source and of the leaves, then the reliability of each task is still matched, and so is the execution time. Moreover, the energy consumption is equal to the energy consumption of the source plus n times the energy consumption of any leaf (recall that they all consume the same amount of energy). Hence, if the energy consumption of the source is smaller than the one of the leaves, permuting those execution speeds would reduce by $(n-1) \times \Delta$ the energy, where Δ is the positive difference between the two energy consumptions. Thanks to this result, we can say that the source should never be executed twice if the leaves are executed only once since it would mean a lower energy consumption for the source (recall that $n \geq 2$).

We have now fully characterized the shape of any optimal solution. There are only three possibilities:

1. no task is re-executed;
2. the source is executed once and the successors (leaves) are re-executed;
3. each task is re-executed.

Next, we study independently the three cases, i.e., we aim at determining the values of $f_{\texttt{src}}$ and $f_{\texttt{leaf}}$ in each case. We will then give conditions on the deadline that indicate what the shape of the solution should be. First we introduce some notations: δ_s is the number of times that the source is executed ($\delta_s = 1$ or $\delta_s = 2$), and $f_{\texttt{src}}$ is its execution speed(s). Similarly, δ_l is the number of times that the leaves are executed, and $f_{\texttt{leaf}}$ their execution speeds. Note that with the previous results, $\delta_l \geq \delta_s$. With these notations the problem we want to minimize becomes:

$$\begin{array}{lll} \text{Minimize} & & w\delta_s f_{\texttt{src}}^2 + n \times w\delta_l f_{\texttt{leaf}}^2 \\ \text{subject to} & \text{(i)} & \delta_s \frac{w}{f_{\texttt{src}}} + \delta_l \frac{w}{f_{\texttt{leaf}}} \leq D, \\ & \text{(ii)} & 1 - \lambda_0^{\delta_s} \frac{w^{\delta_s}}{f_{\texttt{src}}^{\delta_s}} e^{-\delta_s f_{\texttt{src}}} \geq 1 - \lambda_0 \frac{w}{f_{\texttt{rel}}} e^{-df_{\texttt{rel}}}, \\ & \text{(iii)} & 1 - \lambda_0^{\delta_l} \frac{w^{\delta_l}}{f_{\texttt{leaf}}^{\delta_l}} e^{-\delta_l f_{\texttt{leaf}}} \geq 1 - \lambda_0 \frac{w}{f_{\texttt{rel}}} e^{-df_{\texttt{rel}}}. \end{array} \tag{6.9}$$

1. No task is re-executed. Let us assume first that the optimal solution is such that each task is executed only once. From the proof of Theorem 5.1, we obtain the optimal speeds with no re-execution; they are given by the following formulas:

- if $D < \frac{2w}{f_{\max}}$ then there is no solution;
- if $\frac{2w}{f_{\max}} \leq D \leq \frac{w}{f_{\max}}(1 + n^{\frac{1}{3}})$, then $f_{\texttt{src}} = f_{\max}$ and $f_{\texttt{leaf}} = \frac{w}{Df_{\max} - w} f_{\max}$;
- if $\frac{w}{f_{\max}}(1 + n^{\frac{1}{3}}) < D$, then $f_{\texttt{src}} = \frac{w}{D}(1 + n^{\frac{1}{3}})$ and $f_{\texttt{leaf}} = \frac{w}{D} \frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$.

Because there is a minimum speed $f_{\texttt{rel}}$ to match the reliability, there is a condition when $f_{\texttt{leaf}} < f_{\texttt{rel}}$ which makes an amendment on the last item:

- if $\frac{w}{f_{\texttt{rel}}} \frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}} < D \leq \frac{2w}{f_{\texttt{rel}}}$, then $f_{\texttt{src}} = \frac{w}{Df_{\texttt{rel}} - w} f_{\texttt{rel}}$ and $f_{\texttt{leaf}} = f_{\texttt{rel}}$;
- if $\frac{2w}{f_{\texttt{rel}}} < D$, then $f_{\texttt{src}} = f_{\texttt{leaf}} = f_{\texttt{rel}}$.

2. The source is executed once and the leaves are re-executed. Assume now that all leaves are re-executed. We can consider an equivalent DAG where leaves are of weight $2w$, and a schedule with no re-execution. Then the optimal solution when there is no maximum speed is: $f_{\texttt{src}} = \frac{w}{D}(1 + 2n^{\frac{1}{3}})$ and $f_{\texttt{leaf}} = \frac{w}{D} \frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$. Note that if $f_{\texttt{leaf}} \geq \frac{1}{\sqrt{2}} f_{\texttt{rel}}$, then there is a better solution without re-execution (or there is no solution). Indeed, the solution where the leaves are executed once at speed $\max(\frac{w}{D} \frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}, f_{\texttt{rel}})$ is a solution, matches the reliability obviously, the deadline ($\frac{w}{\max(\frac{w}{D} \frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}, f_{\texttt{rel}})} < \frac{2\sqrt{2}w}{f_{\texttt{rel}}} \leq \frac{2w}{\frac{w}{D} \frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}}$), and it has a better energy consumption (because $\frac{w}{D} \frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}} \geq \frac{1}{\sqrt{2}} f_{\texttt{rel}}$).

Since we are in case 2 with re-execution of the leaves, we can assume that $\frac{w}{f_{\texttt{rel}}} \sqrt{2} \frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}} < D$. Then, depending if $f_{\texttt{src}} \geq f_{\texttt{rel}}$ or $f_{\texttt{src}} < f_{\texttt{rel}}$:

- if $f_{\texttt{src}} \geq f_{\texttt{rel}}$, it means that $D \leq \frac{w}{f_{\texttt{rel}}}(1 + 2n^{\frac{1}{3}})$, ($\frac{w}{f_{\texttt{rel}}}(1 + 2n^{\frac{1}{3}}) > \frac{w}{f_{\texttt{rel}}} \sqrt{2} \frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$ is true when $n > 2$), then $f_{\texttt{src}} = \frac{w}{D}(1 + 2n^{\frac{1}{3}})$ and $f_{\texttt{leaf}} = \frac{w}{D} \frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$;

- if $f_{\texttt{src}} < f_{\texttt{rel}}$, it means that $D > \frac{w}{f_{\texttt{rel}}}(1+2n^{\frac{1}{3}})$, then $f_{\texttt{src}} = f_{\texttt{rel}}$ and $f_{\texttt{leaf}} = \max(\frac{2w}{Df_{\texttt{rel}}-w} f_{\texttt{rel}}, f_{\min})$.

3. Each task is re-executed. If the solution is such that each task is re-executed (source or leaf), it is equivalent to consider a DAG with tasks of weight $2w$ and no re-execution. Then the optimal solution when there is no maximum speed is: $f_{\texttt{src}} = \frac{2w}{D}(1+n^{\frac{1}{3}})$ and $f_{\texttt{leaf}} = \frac{2w}{D}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$. Note that if $f_{\texttt{src}} \geq \frac{1}{\sqrt{2}} f_{\texttt{rel}}$, then there is a better solution in which the source is not re-executed. Indeed, the solution where the source is executed once at speed $\max(\frac{2w}{D}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}}, f_{\texttt{rel}})$ is a solution, matches the reliability obviously, the deadline ($\frac{w}{\max(\frac{2w}{D}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}}, f_{\texttt{rel}})} < \frac{2\sqrt{2}w}{f_{\texttt{rel}}} \leq \frac{2w}{\frac{2w}{D}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}}}$) and has a better energy consumption (because $\frac{2w}{D}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}} \geq \frac{1}{\sqrt{2}} f_{\texttt{rel}}$).

Therefore, if $D \leq \frac{w}{f_{\texttt{rel}}} 2\sqrt{2}(1+n^{\frac{1}{3}})$, the source should not be re-executed.

No re-execution $\rightarrow$ leaves re-executed. To complete the proof, there remains to establish the value of the deadline at which re-execution will be used by the optimal solution, i.e., at which point we will move from case 1 to case 2. We know that the minimum energy consumption is a function decreasing with the deadline: if $D > D'$, then any solution for D' is a solution for D. Let us find the minimum deadline D such that the energy when the leaves are re-executed is smaller than the energy when no task is re-executed.

As we have seen before, necessarily if $D \leq \frac{w}{f_{\texttt{rel}}}\sqrt{2}\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$, then it is better to have no re-execution. Let $D = \frac{w}{f_{\texttt{rel}}}\sqrt{2}\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}} + \varepsilon$. Suppose first that $D \leq \frac{w}{f_{\texttt{rel}}}(1+2n^{\frac{1}{3}})$, then the energy when the leaves are re-executed is: $E_2 = \frac{w^3}{D^2}(1+2n^{\frac{1}{3}})^3$. With no re-execution the total energy is: $E_1 = (1+n)w f_{\texttt{rel}}^2 = 2\frac{w^3}{(D-\varepsilon)^2}(1+n)\left(\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}\right)^2$.

We now check the condition $E_1 \geq E_2$:

$$
\begin{aligned}
2\frac{w^3}{(D-\varepsilon)^2}(1+n)\left(\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}\right)^2 &\geq \frac{w^3}{D^2}(1+2n^{\frac{1}{3}})^3 \\
\frac{2}{(D-\varepsilon)^2}\frac{1+n}{n^{\frac{2}{3}}} &\geq \frac{1+2n^{\frac{1}{3}}}{D^2} \\
\frac{D^2}{(D-\varepsilon)^2} &\geq \frac{n^{\frac{2}{3}}+2n}{2+2n} \\
D &\geq \frac{w}{f_{\texttt{rel}}}\frac{(1+2n^{\frac{1}{3}})^{\frac{3}{2}}}{\sqrt{1+n}}
\end{aligned}
$$

Furthermore, the set $\frac{w}{f_{\texttt{rel}}}\frac{(1+2n^{\frac{1}{3}})^{\frac{3}{2}}}{\sqrt{1+n}} \leq D \leq \frac{w}{f_{\texttt{rel}}}(1+2n^{\frac{1}{3}})$ is not empty when $n > 2$.

This completes the proof for the boundary between cases 1 and 2: if the deadline is smaller than the threshold value $\frac{w}{f_{\texttt{rel}}}\frac{(1+2n^{\frac{1}{3}})^{\frac{3}{2}}}{\sqrt{1+n}}$, the optimal solution will not do any re-execution. However, if the deadline is larger, then it is better to re-execute the leaves, and hence the optimal solution will be in the shape of case 2.

Source executed once → source re-executed. As we have seen earlier, when $D \leq \frac{w}{f_{\texttt{rel}}} 2\sqrt{2}(1+n^{\frac{1}{3}})$, the source should not be re-executed, and therefore the optimal solution will be in the shape of case 1 or 2. As stated in the proposition, the cases with larger deadlines are not fully developed here. Hence the proof is complete. ■

Beyond the proof itself, the result of Proposition 6.2 is interesting: we observe that in all cases, the source task is executed faster than the other tasks. This shows that Proposition 6.1 does not hold for general DAGs, and suggests that some tasks may be more critical than others. A hierarchical approach, that categorizes tasks with different priorities, will guide the design of type B heuristics in Section 6.4.

6.4 Heuristics for TRI-CRIT-CONT

In this section, we propose some polynomial-time heuristics for TRI-CRIT-CONT, which was shown NP-hard (see Theorem 6.1). We start by outlining the general principles that have guided the design before exposing the details for each heuristic.

6.4.1 General principles

The heuristics work in two steps: first we apply a simple list scheduling algorithm in order to map the DAG onto the p processors. More precisely, we apply a critical-path list scheduling, which assigns the most urgent ready task (with largest bottom-level) to the first available processor. The bottom-level is defined as $bl(T_i) = w_i$ if T_i has no successor task, and $bl(T_i) = w_i + \max_{(T_i,T_j)\in\mathcal{E}} bl(T_j)$ otherwise. At the end of this first step, each task has been mapped on a processor, and it is scheduled for a single execution at maximum speed.

Then, the core of the heuristics consists in reducing the energy consumption of the schedule. We do not change the allocation of the tasks during this second step, but we can slow them down and/or re-execute them.

The first energy reduction technique comes from Proposition 6.1. The idea is to start by searching for the optimal solution of the problem instance without re-execution, a phase that we call *deceleration*: here we slow down some tasks if it can save energy without violating one of the constraints. Then we refine the schedule and choose the tasks that we want to re-execute, according to some criteria. We call *type A heuristics* such heuristics that obey this general scheme: first deceleration then re-execution. Type A heuristics are expected to be efficient on a DAG with a low degree of parallelism (optimal for a chain). However, Proposition 6.2 (with fork graphs) shows that it might be better to re-execute the highly parallel tasks before decelerating. Therefore we introduce *type B heuristics*, which first choose the set of tasks to be re-executed, and then try to slow down the tasks that could not be re-executed. We need to find good criteria to select tasks to be re-executed, so that type B heuristics prove efficient for DAGs with a high degree of parallelism. In summary, type B heuristics obey the opposite scheme: first re-execution then deceleration.

For both heuristic types, the approach for each phase can be sketched as follows. Let r_i be the start time of task T_i in the current configuration, and d_i be its finish time.

Deceleration: We select a set of tasks that we execute at speed $\max(f_{\texttt{rel}}, \frac{\max_{i=1..n} d_i}{D} f_{\max})$, which is the slowest possible speed meeting both the reliability and deadline constraints.

Re-execution: We greedily select tasks for re-execution. The selection criterion is either by decreasing weights w_i, or by decreasing *super-weights* W_i. The super-weight of a task T_i is defined as the sum of the weights of the tasks (including T_i) whose execution interval is included into T_i's execution

interval. The rationale is that the super-weight of a task that we slow down is an estimation of the total amount of work that can be slowed down together with that task, hence of the energy potentially saved: this corresponds to the total slack that can be reclaimed.

6.4.2 List of heuristics

In this section, we describe the different energy-reducing heuristics in more details. We first introduce a few notations:

- *SUS* (Slack-Usage-Sort) is a function that sorts tasks by decreasing super-weights.
- *ReExec* is a function that tries to re-execute the current task T_i, at speed $f_{\texttt{re-ex}} = \frac{2c}{1+c} f_{\texttt{rel}}$ (note that $f_{\texttt{re-ex}}$ is the optimal speed in the proof of Theorem 6.1). If it succeeds, it also re-executes at speed $f_{\texttt{re-ex}}$ all the tasks that are taken into account to compute the super-weight of T_i. Otherwise, it does nothing.
- *ReExec&SlowDown* performs the same re-executions as *ReExec* when it succeeds. But if the re-execution of the current task T_i is not possible, it slows down T_i as much as possible and does the same for all the tasks that are taken into account to compute the super-weight of T_i.

We now detail the heuristics:

H$f_{\max}$. In this heuristic, tasks are simply executed once and at maximum speed, by the processor assigned to them by the list scheduling heuristic.

Hno-reex. In this heuristic, we do not allow any re-execution, and we simply consider the possible deceleration of the tasks. We set a uniform speed for all tasks, equal to $\max(f_{\texttt{rel}}, \frac{\max_{i=1..n} d_i}{D} f_{\max})$, so that both the reliability and deadline constraints are matched.

A.Greedy. This is a type A heuristic, where we first set the speed of each task to $\max(f_{\texttt{rel}}, \frac{\max_{i=1..n} d_i}{D} f_{\max})$, so that both the reliability and deadline constraints are matched (deceleration). Let Greedy-List be the list of all the tasks sorted according to decreasing weights w_i. Each task T_i in Greedy-List is re-executed at speed $f_{\texttt{re-ex}}$ whenever possible. Finally, if there remains some slack at the end of the processing, we slow down both executions of each re-executed task as much as possible.

A.SUS-Crit. This is a type A heuristic, where we first set the speed of each task to $\max(f_{\texttt{rel}}, \frac{\max_{i=1..n} d_i}{D} f_{\max})$, just as in the previous heuristic. Let List-SW be the list of all tasks that belong to a critical path, sorted according to SUS. We apply ReExec to List-SW (re-execution). Finally we reclaim slack for re-executed tasks, similarly to the final step of **A.Greedy**.

B.Greedy. This is a type B heuristic. We use Greedy-List as in heuristic **A.Greedy**. We try to re-execute each task T_i of Greedy-List when possible. Then, we slow down both executions of each re-executed task T_i of Greedy-List as much as possible. Finally, we slow down the speed of each task of Greedy-List that turn out not re-executed, as much as possible.

B.SUS-Crit. This is a type B heuristic. We use List-SW as in heuristic **A.SUS-Crit**. We apply ReExec to List-SW (re-execution). Then we run Heuristic B.Greedy.

B.SUS-Crit-Slow. This is a type B heuristic. We use List-SW, and we apply ReExec&SlowDown (re-execution). Then we use Greedy-List: for each task T_i of Greedy-List, if there is enough time, we execute twice T_i at speed $f_{\texttt{re-ex}}$ (re-execution); otherwise, we execute T_i only once, at the slowest admissible speed.

Best. This is simply the minimum value over the six previous heuristics, for reference.

The complexity of all these heuristics is bounded by $O(n^4 \log n)$. The most time-consuming operation is the computation of List-SW (the list of all elements that belong to a critical path, sorted according to SUS).

6.5 Simulations

In this section, we report extensive simulations to assess the performance of the heuristics presented in Section 6.4. All the source-code (our heuristics were coded using the programming language OCaml), together with additional results that were omitted due to lack of space, are publicly available at [4].

6.5.1 Simulation settings

In order to evaluate the heuristics, we have generated DAGs using the random DAG generation library GGEN [33]. Since GGEN does not give a weight to the tasks of the DAGs, we use a function that gives a random float value in the interval $[0, 10]$. Each simulation uses a DAG with 100 nodes and 300 edges. We observe similar patterns for other number of edges, see [4] for further information. We choose a reliability constant $\lambda_0 = 10^{-5}$ [3]. We obtain identical results when λ_0 varies from 10^{-5} to 10^{-6} (see Figure 6.5). Each reported result is the average on ten different DAGs with the same number of nodes and edges, and the energy consumption is normalized with the energy consumption returned by the **Hno-reex** heuristic. If the value is lower than 1, it means that we have been able to save energy thanks to re-execution.

We analyze the influence of three different parameters: the tightness of the deadline D, the processor number p and the reliability speed $f_{\texttt{rel}}$. In fact, the absolute deadline D is irrelevant, and we rather consider the *deadline ratio*:

$$\text{DEADLINERATIO} = \frac{D}{D_{\texttt{min}}},$$

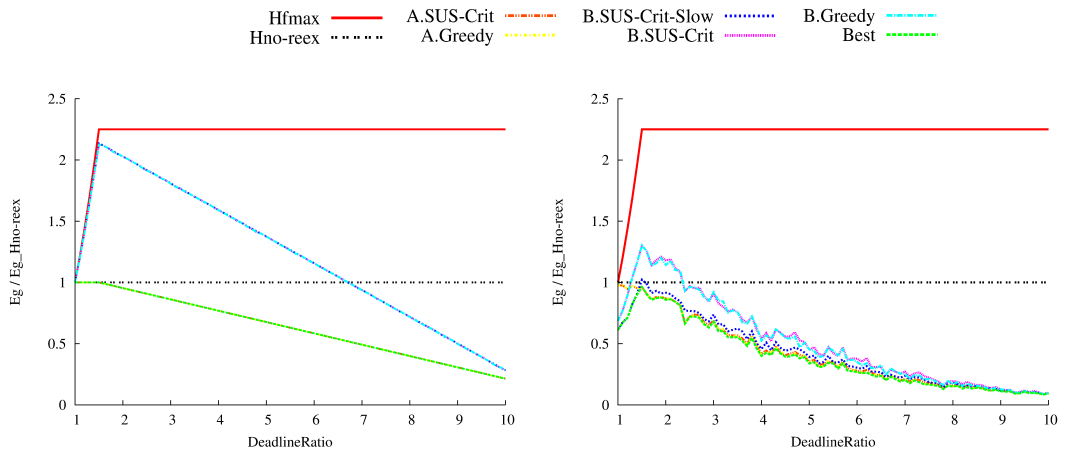
where $D_{\texttt{min}}$ is the execution time of the list scheduling heuristic when executing each task once and at speed $f_{\max}$. Intuitively, when the deadline ratio is close to 1, there is almost no flexibility and it is difficult to re-execute tasks, while when the deadline ratio is larger we expect to be able to slow down and re-execute many tasks, thereby saving much more energy.

6.5.2 Simulation results

We first note that when there is only one processor, heuristics A.SUS-Crit and A.Greedy are identical, and heuristics B.SUS-Crit and B.Greedy are identical (by definition, the only critical path is the whole set of tasks).

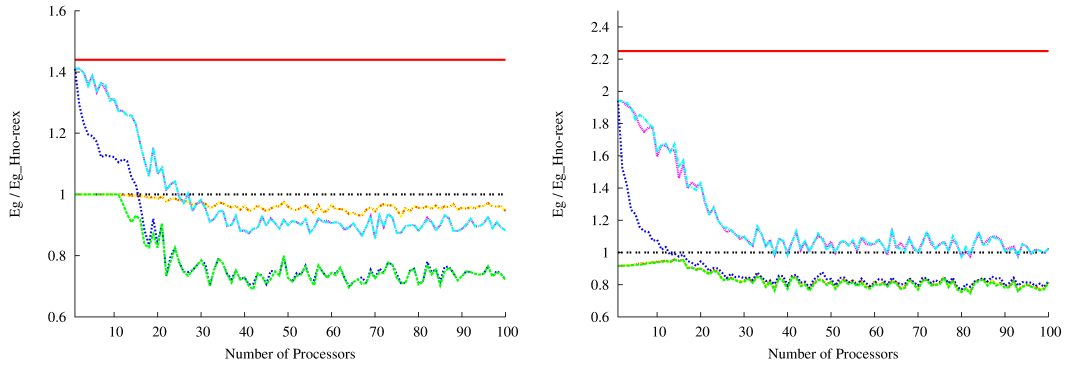
Deadline ratio

In this set of simulations, we let $p \in \{1, 10, 50, 70\}$ and $f_{\texttt{rel}} = \frac{2}{3} f_{\max}$. Figure 6.2 reports results for $p = 1$ and $p = 50$. When $p = 1$, we see that the results are identical for all heuristics of type A, and identical for all heuristics of type B. As expected from Proposition 6.1, type A heuristics are better (see Figure 6.2a). With more processors (10, 50, 70), the results have the same general shape: see Figure 6.2b with 50 processors. When DEADLINERATIO is small, type B heuristics are better. When DEADLINERATIO increases up to 1.5, type A heuristics are closer to type B ones. Finally, when DEADLINERATIO gets larger than 5, all heuristics converge towards the same result, where all tasks are re-executed.



(a) 1 processor (b) 50 processors

Figure 6.2: Comparative study when the deadline ratio varies.



(a) DEADLINERATIO = 1.2 (b) DEADLINERATIO = 2.4

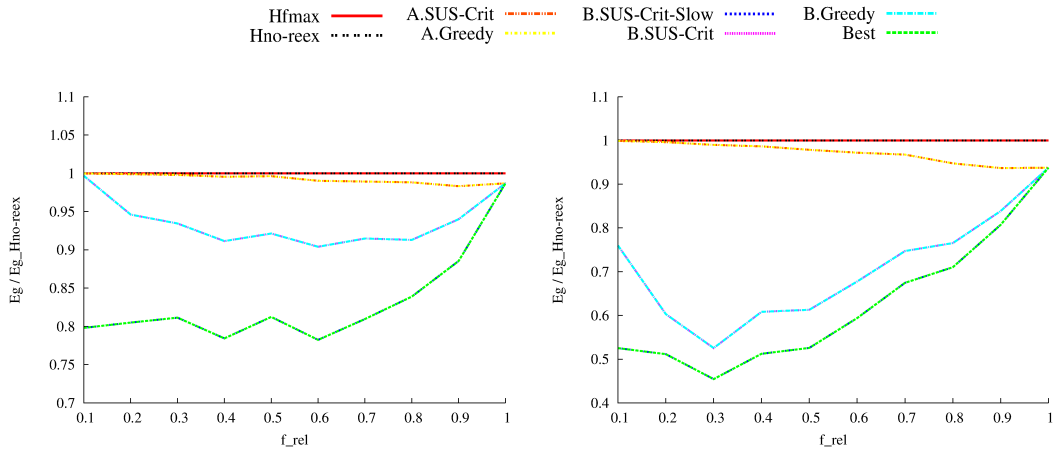
Figure 6.3: Comparative study when the number of processors p varies.

Number of processors

In this set of simulations, we let DEADLINERATIO $\in \{1.2, 1.6, 2, 2.4\}$ and $f_{\texttt{rel}} = \frac{2}{3} f_{\max}$. Figure 6.3 confirms that type A heuristics are particularly efficient when the number of processors is small, whereas type B heuristics are at their best when the number of processors is large. Figure 6.3a confirms the superiority of type B heuristics for tight deadlines, as was observed in Figure 6.2b.

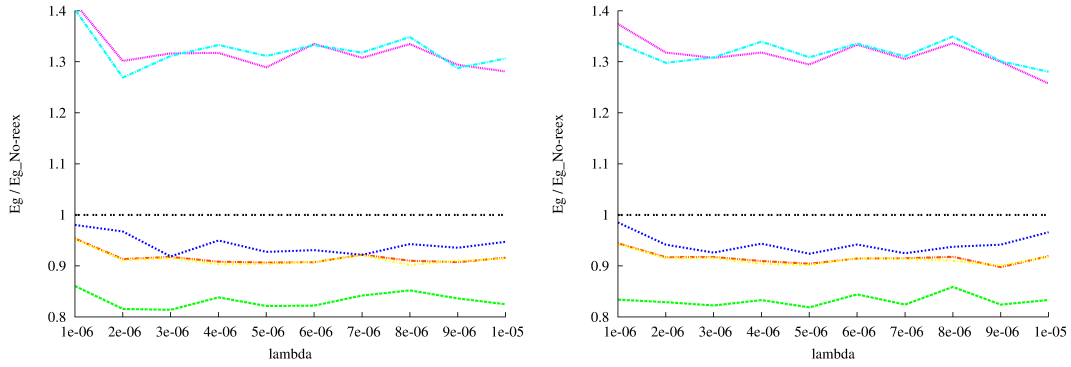
Reliability $f_{\texttt{rel}}$

In this set of simulations, we let $p \in \{1, 10, 50, 70\}$ and DEADLINERATIO $\in \{1, 1.5, 3\}$. In Figure 6.4, there are four different curves: the line at 1 corresponds to Hno-reex and H$f_{\max}$, then come the heuristics of type A (that all obtain exactly the same results), then B.SUS-Crit and B.Greedy that also obtain the same results, and finally the best heuristic is B.SUS-Crit-Slow. Note that B.SUS-Crit and B.Greedy return the same results because they have the same behavior when DEADLINERATIO $= 1$: there is no



(a) DEADLINERATIO = 1, 10 processors (b) DEADLINERATIO = 1, 50 processors

Figure 6.4: Comparative study when the reliability $f_{\texttt{rel}}$ varies.



(a) DEADLINERATIO = 1.1, 50 processors (b) DEADLINERATIO = 1.5, 50 processors

Figure 6.5: Comparative study when the λ_0 varies.

liberty of action on the critical paths. However B.SUS-Crit-Slow gives better results due to its ability of slowing more tasks down. When DEADLINERATIO is really tight (equal to 1), decreasing the value of $f_{\texttt{rel}}$ from $f_{\texttt{rel}} = 1$ down to $f_{\texttt{rel}} = 0.9$ makes a real difference with type B heuristics. We observe an energy gain of 10% when the number of processors is small (10 in Figure 6.4a) and of 20% with more processors (50 in Figure6.4b).

6.5.3 Understanding the results

A.SUS-Crit and A.Greedy, and B.SUS-Crit and B.Greedy, often obtain similar results, which might lead us to underestimate the importance of critical path tasks. However, the difference between B.SUS-Crit-Slow and B.SUS-Crit shows otherwise. Tasks that belong to a critical path must be dealt with first.

A striking result is the impact of both the number of processors and the deadline ratio on the effectiveness of the heuristics. Heuristics of type A, as suggested by Proposition 6.1, have clearly better results when there is a small number of processors. When the number of processors increases, there is a

difference between small deadline ratio and larger deadline ratio. In particular, when the deadline ratio is small, heuristics of type B will have better results. Here is an explanation: heuristics of type A try to accommodate as many tasks as possible, and as a consequence, no task can be re-executed. On the contrary, heuristics of type B try to favor some tasks that are considered as important. This is highly profitable when the deadline is tight.

Altogether we have identified two very efficient and complementary heuristics, A.SUS-Crit and B.SUS-Crit-Slow. Taking the best result out of those two heuristics always gives the best result over all simulations.

6.6 Conclusion

In this chapter, we have introduced a new energy model for re-execution, that seems more realistic and accurate than the best-case model used by Zhu et al. [133]. Coupling this model with the classical reliability model presented by Shatz and Wang [113], we have been able to formulate a tri-criteria optimization problem: how to minimize the energy consumption given a deadline bound and a reliability constraint? We have assessed the intractability of this tri-criteria problem, even when the mapping of tasks to processors is already known. In addition, we have provided several complexity results for particular instances.

We have designed and evaluated some polynomial-time heuristics for the TRI-CRIT-CONT problem that are based on the failure probability, the task weights, and the processor speeds. These heuristics aim at minimizing the energy consumption of the list-schedule execution of the application while enforcing reliability and deadline constraints. They rely on *dynamic voltage and frequency scaling* (DVFS) to decrease the energy consumption. But because DVFS lowers the reliability of the system, the heuristics use *re-execution* to compensate for the loss. After running several heuristics on a wide class of problem instances, we have identified two heuristics that are complementary, and that together are able to produce good results on most instances.

All the heuristics slow down or re-execute tasks without changing their assignment to processors. In other words, they do not modify the mapping determined by the list-schedule, hence they can also be used when the mapping of the application is already given, e.g. for affinities or security reasons [103].

Future work involves several promising directions. On the theoretical side, it would be very interesting to prove a competitive ratio for the heuristic that takes the best out of A.SUS-Crit and B.SUS-Crit-Slow. However, this is quite a challenging work for arbitrary DAGs, and one may try to design approximation algorithms only for special graph structures, e.g. series-parallel graphs. Also, it would be important to assess the impact of the list scheduling heuristic that precedes the energy-reduction heuristic. In other words, the classical critical-path list-scheduling heuristic, which is known to be efficient for deadline minimization, may well be superseded by another heuristic that trades-off execution time, energy and reliability when mapping ready tasks to processors. Such a study could open new avenues for the design of multi-criteria list-scheduling heuristics.

Finally, we point out that energy reduction and reliability will be even more important objectives with the advent of massively parallel platforms, made of a large number of clusters of multi-cores. More efficient solutions to the tri-criteria optimization problem (deadline, energy, reliability) could be achieved through combining replication with re-execution. A promising (and ambitious) research direction would be to search for the best trade-offs that can be achieved between these techniques that both increase reliability, but whose impact on execution time and energy consumption is very different.

Chapter 7

Approximation algorithms for energy, reliability and makespan optimization problems

7.1 Introduction

In Chapter 6, we considered, motivated by the application of speed scaling on large scale machines [96], a tri-criteria problem energy/reliability/makespan: the goal was to minimize the energy consumption, while enforcing a bound on the makespan, i.e., the total execution time, and a constraint on the reliability of each task. In order to make up for the loss in reliability due to the energy efficiency, we considered in the previous chapter the standard *re-execution* technique, which consists in re-executing a task twice on the same processor [134, 133]. In this chapter, we further add the *replication* technique, which consists in executing the same task on two distinct processors simultaneously [3].

In this chapter, we restrict the study to specific applications: either a linear chain of tasks, or a set of independent tasks. The platform is made of identical processors, whose speed can be dynamically modified. The schedule therefore requires us to (i) decide which tasks are re-executed or replicated; (ii) decide on which processor(s) each task is executed; (iii) decide at which speed each processor is processing each task. For a given schedule, we can compute the total execution time, also called *makespan*, and it should not exceed a prescribed deadline. Each task has a reliability that can be computed given its execution speed and its eventual replication or re-execution, and we must enforce that the execution of each task is reliable enough. Finally, we aim at minimizing the energy consumption. Note that we consider a set of homogeneous processors, but each processor may run at a different speed; this corresponds to typical current platforms with DVFS.

Main contributions. In this chapter, we investigate the tri-criteria problem of minimizing the energy consumption with a bound on the makespan and a constraint on the reliability. First in Section 7.2, we formally introduce this tri-criteria scheduling problem, based on the previous models proposed by Zhu and Aydin [133] and used in Chapter 6. To the best of our knowledge, this is the first model including both re-execution and replication in order to deal with failures. The main contribution of this chapter is then to provide approximation algorithms for some particular instances of this tri-criteria problem.

For linear chains of tasks, we propose a fully polynomial-time approximation scheme (Section 7.3). Then in Section 7.4, we show that there exists no constant factor approximation algorithm for the tri-criteria problem with independent tasks, unless P=NP. We prove that by relaxing the constraint on the makespan, we can give a polynomial-time constant factor approximation algorithm. To the best of our

knowledge, these are the first approximation algorithms for the tri-criteria problem.

7.2 Framework

p	Number of processors in the platform
$\mathcal{G} = (V, \mathcal{E})$	Application graph
T_i	i^{th} task
w_i	Weight of T_i
$S = \sum_i w_i$	Sum of the weight of all tasks of $\mathcal{G}$
$f_{\min}$	Minimum available speed
$f_{\max}$	Maximum available speed
$f_{\texttt{rel}}$	Minimum speed so that a single execution reaches the reliability constraint
$f_{\inf,i}$	Minimum speed not lower than $f_{\min}$, so that two executions of T_i at that speed reach the reliability constraint
$\mathcal{E}xe(w_i, f_i)$	Execution time of a task of weight w_i at speed f_i
D	Deadline at which the execution should be completed

Table 7.1: Table of main notations.

We use the same model as presented in Chapter 6. We summarize the main notations of this chapter in Table 7.1. Consider an application task graph $\mathcal{G} = (V, \mathcal{E})$, where the vertex set $V = \{T_1, \ldots, T_n\}$ is the set of n tasks, and the edge set $\mathcal{E}$ corresponds to the precedence constraints between tasks. For $1 \leq i \leq n$, task T_i has a weight w_i, that corresponds to the computation requirement of the task. $S = \sum_{i=1}^{n} w_i$ is the sum of the computation requirements of all tasks.

The goal is to map the task graph onto p identical processors that can have arbitrary speeds, determined by their frequency, that can take any value in the interval $[f_{\min}, f_{\max}]$ (dynamic voltage and frequency scaling with continuous speeds). Higher frequencies, and hence faster speeds, allow for a faster execution, but they also lead to a much higher (supra-linear) power consumption. Note that we showed in Lemma 6.1 that it is always better to execute a task at a single speed, and therefore we assume in the following that each execution of a task is done at a single speed.

We now detail the three objective criteria (makespan, reliability, energy), and then formally define the optimization problem in Section 7.2.4.

7.2.1 Makespan

The makespan of a schedule is its total execution time. The first task is scheduled at time 0, so that the makespan of a schedule is simply the maximum time at which one of the processors finishes its computations. Given a schedule, the makespan should not exceed the prescribed deadline D.

Let $\mathcal{E}xe(w_i, f)$ be the execution time of a task T_i of weight w_i at speed f. We enforce the classical linear cost model for execution times [85]: $\mathcal{E}xe(w_i, f) = \frac{w_i}{f}$. Note that we consider a worst-case scenario, and the deadline D must be matched even in the case where all tasks that are scheduled to be executed several times fail during their first executions, hence all execution and re-execution times should be accounted for.

7.2.2 Reliability

We consider the fault-model introduced in Section 6.2.2. The reliability of a task T_i executed once at speed f is the probability of a successful execution, and it is expressed as

$$\begin{aligned} R_i(f) &\approx 1 - \lambda(f) \times \mathcal{E}xe(w_i, f) \\ &= 1 - \tilde{\lambda_0}\, e^{\tilde{d}\frac{f_{\max}-f}{f_{\max}-f_{\min}}} \times \frac{w_i}{f} \\ &= 1 - \lambda_0\, e^{-df} \times \frac{w_i}{f}, \end{aligned}$$

Note that this equation holds if $\lambda(f) \times \frac{w_i}{f} \ll 1$. With, say, $\lambda(f) = 10^{-5}$, we need $\frac{w_i}{f} \leq 10^3$ to get an accurate approximation with $\lambda(f) \times \frac{w_i}{f} \leq 0.01$: the task should execute within 16 minutes. In other words, large (computationally demanding) tasks require reasonably high processing speeds with this model (which makes full sense in practice).

As in Chapter 6, we consider that a task is reliable enough when it is executed once at a speed greater than or equal to a threshold speed $f_{\texttt{rel}} = \alpha f_{\max}$, where $\frac{f_{\min}}{f_{\max}} \leq \alpha \leq 1$ is fixed by the user and corresponds to the reliability of the system. For highly critical systems, $\alpha = 1$ and therefore $f_{\texttt{rel}} = f_{\max}$ [132]. In order to limit energy consumption, the execution speed of a task can be further decreased, but then the probability of having at least one transient failure during the execution of this task increases drastically, both because of the extended execution time and the increased failure rate $\lambda(f)$. In this case, we therefore enforce the execution of a *backup task* [133, 132]. We do not execute automatically this task at the maximum speed (or speed $f_{\texttt{rel}}$) as was done in previous work, but rather we choose a re-execution speed such that the reliability of both executions is at least equal to the reliability of a single execution at speed $f_{\texttt{rel}}$. Therefore, either task T_i is executed only once at speed $f \geq f_{\texttt{rel}}$, or it is executed twice (speeds $f^{(1)}$ and $f^{(2)}$), and the reliability, i.e., the probability that at least one of the attempts do not fail: $R_i = 1 - (1 - R_i(f^{(1)}))(1 - R_i(f^{(2)}))$ should be at least equal to $R_i(f_{\texttt{rel}})$.

We restrict to one single backup task, which can be scheduled either on the same processor as the original task (what we call *re-execution*), or on another processor (what we call *replication*). Intuitively, having two or more backup tasks may lead to further energy savings, but at a price of a highly increased execution time (and a much more complex study).

Note that if both execution speeds are equal, i.e., $f^{(1)} = f^{(2)} = f$, then the reliability constraint becomes $1 - (\lambda_0 w_i \frac{e^{-df}}{f})^2 \geq R_i(f_{\texttt{rel}})$, and therefore

$$\lambda_0 w_i \frac{e^{-2df}}{f^2} \leq \frac{e^{-df_{\texttt{rel}}}}{f_{\texttt{rel}}}\ .$$

In the following, $f_{\text{inf},i}$ is the maximum between $f_{\min}$ and the solution to the equation $\lambda_0 w_i \frac{e^{-2df_{\text{inf},i}}}{(f_{\text{inf},i})^2} = \frac{e^{-df_{\texttt{rel}}}}{f_{\texttt{rel}}}$, and hence if task T_i is executed twice at a speed greater than or equal to $f_{\text{inf},i}$, then the reliability constraint is met.

7.2.3 Energy

The total energy consumption corresponds to the sum of the energy consumption of each task. Let E_i be the energy consumed by task T_i. For one execution of T_i at speed f, the corresponding energy consumption is $E_i(f) = \mathcal{E}xe(w_i, f) \times f^3 = w_i \times f^2$, which corresponds to the dynamic part of the classical energy models of the literature [6, 8]. Note that we do not take static energy into account, because all processors are up and alive during the whole execution.

If task T_i is executed only once at speed f, then $E_i = E_i(f)$. Otherwise, if task T_i is executed twice at speeds $f^{(1)}$ and $f^{(2)}$, it is natural to add up the energy consumed during both executions, just as we consider both execution times when enforcing the deadline on the makespan. Again, this corresponds to the worst-case execution scenario. We obtain $E_i = E_i(f_i^{(1)}) + E_i(f_i^{(2)})$. Finally, the total energy consumed by the schedule, which we aim at minimizing, is $E = \sum_{i=1}^{n} E_i$.

7.2.4 Optimization problem

Given an application graph $\mathcal{G} = (V, \mathcal{E})$ and p identical processors, TRI-CRIT is the problem of finding a schedule that specifies which tasks should be executed twice, on which processor and at which speed each execution of a task should be processed, such that the total energy consumption E is minimized, subject to the deadline D on the makespan and to the local reliability constraints $R_i \geq R_i(f_{\texttt{rel}})$ for each $T_i \in V$.

Note that TRI-CRIT may have no solution: it may well be the case that the deadline cannot be enforced even if all tasks are executed only once at speed $f_{\max}$.

We focus in this chapter on the two following sub-problems that are restrictions of TRI-CRIT to special application graphs:

- TRI-CRIT-CHAIN: the graph is such that $\mathcal{E} = \cup_{i=1}^{n-1} \{T_i \rightarrow T_{i+1}\}$;
- TRI-CRIT-INDEP: the graph is such that $\mathcal{E} = \emptyset$.

7.3 Linear chains

In this section, we focus on the TRI-CRIT-CHAIN problem, that was shown to be NP-hard even on a single processor (Theorem 6.1). We derive an FPTAS (Fully Polynomial-Time Approximation Scheme) to solve the general problem with replication and re-execution on p processors. We start with some preliminaries in Section 7.3.1 that allow us to characterize the shape of an optimal solution, and then we detail the FPTAS algorithm and its proof in Section 7.3.2.

Note that TRI-CRIT-CHAIN has a solution if and only if $\frac{S}{f_{\max}} \leq D$: all tasks must fit within the deadline when executed at the maximum speed. In this section, we therefore assume that $\frac{S}{f_{\max}} \leq D$, otherwise there is no solution.

7.3.1 Characterization

While TRI-CRIT-CHAIN is NP-hard even on a single processor, the problem has polynomial complexity if no replication nor re-execution can be used (see Proposition 5.2). Indeed, each task is executed only once, and the energy is minimized when all tasks are running at the same speed.

Lemma 7.1. *Without replication or re-execution, solving* TRI-CRIT-CHAIN *can be done in polynomial time, and each task is executed at speed* $\max\left(f_{\texttt{rel}}, \frac{S}{D}\right)$.

Proof. For a linear chain of tasks, all tasks can be mapped on the same processor, and scheduled following the dependencies. No task may start earlier by using another processor, and all tasks run at the same speed. Since there is no replication nor re-execution, each task must be executed at least at speed $f_{\texttt{rel}}$ for the reliability constraint. If $S/f_{\texttt{rel}} > D$, then the tasks should be executed at speed S/D so that the deadline constraint is matched (recall that $S = \sum_{i=1}^{n} w_i$), hence the result. This is feasible because $S/D \leq f_{\max}$. ■

Next, accounting for replication and re-execution, we characterize the shape of an optimal solution. The result for linear chains is trivial, with a single processor, only re-execution will be used, while with more than two processors, there is an optimal solution that does not use re-execution, but only replication.

Lemma 7.2 (Replication or re-execution)**.** *When there is only one processor, it is optimal to only use re-execution to solve* TRI-CRIT-CHAIN*. When there are at least two processors, it is optimal to only use replication to solve* TRI-CRIT-CHAIN*.*

Proof. With one processor, the result is obvious, since replication cannot be used. With more than one processor, if re-execution was used on task T_i, for $1 \leq i \leq n$, we can derive a solution with the same energy consumption and a smaller execution time by using replication instead of re-execution. Indeed, all instances of tasks T_j, for $j < i$, must finish before T_i starts its execution, and similarly, all instances of tasks T_j, for $j > i$, cannot start before both copies of T_i has finished its execution. Therefore, there are always at least two processors available when executing T_i for the first time, and the execution time is reduced when executing both copies of T_i in parallel (replication) rather than sequentially (re-execution). ■

We further characterize the shape of an optimal solution by showing that two copies of the same task can always be executed at the same speed.

Lemma 7.3 (Speed of the replicas)**.** *For a linear chain, when a task is executed two times, it is optimal to have both replicas executed at the same speed.*

Proof. With one processor, we have seen in the previous lemma that it was optimal to only use re-execution. The proof for re-execution has been done in Proposition 6.2: by convexity of the energy and reliability functions, it is always advantageous to execute two times the task at the same speed, even if the application is not a linear chain.

With two or more processors, we have seen in the previous lemma that it was optimal to only use replication. Let us consider a solution for which there exists i such that task T_i is executed twice at speeds $f_1 < f_2$. Then the solution where task T_i is executed twice at speed $\frac{f_1+f_2}{2}$ meets the reliability and makespan constraints, and has a lower energy consumption.

Reliability

$$
\begin{aligned}
&1-\left(1-R_i\left(\frac{f_1+f_2}{2}\right)\right)^2-(1-(1-R_i(f_1))(1-R_i(f_2)))\\
&=-\left(2\lambda_0 w_i\frac{e^{-d(f_1+f_2)/2}}{f_1+f_2}\right)^2+\left(\lambda_0 w_i\frac{e^{-df_1}}{f_1}\right)\left(\lambda_0 w_i\frac{e^{-df_2}}{f_2}\right)\\
&=\lambda_0^2 w_i^2\left(\frac{e^{-d(f_1+f_2)}}{f_1 f_2}-4\frac{e^{-d(f_1+f_2)}}{(f_1+f_2)^2}\right).
\end{aligned}
$$

This is strictly positive because $(f_1-f_2)^2 = f_1^2+f_2^2-2f_1f_2 > 0$, and therefore $(f_1+f_2)^2 > 4f_1f_2$. Therefore, the reliability of the new solution is greater than the reliability of the solution with distinct speeds.

Makespan The previous execution time of the task was $\frac{w_i}{f_1}$ since $f_1 < f_2$ and both executions are simultaneous (see Proof of Lemma 7.2). It becomes $\frac{2w_i}{f_1+f_2} < \frac{w_i}{f_1}$, and therefore the deadline constraint is still met.

Energy Finally, we show that we have a better energy consumption:

$$2w_i\left(\frac{f_1+f_2}{2}\right)^2 - w_i\left(f_2^2+f_1^2\right) = -\frac{w_i}{2}\left(f_1-f_2\right)^2 < 0.$$

To conclude, we have shown that if we have a solution where a task is executed twice, but both executions are not at the same speed, then we can exhibit a better solution (in terms of energy consumption) that meets the reliability and makespan constraints with one unique speed.

■

We can further characterize an optimal solution by providing detailed information about the execution speeds of the tasks, depending on whether they are executed only once, re-executed, or replicated.

Proposition 7.1. *If $D > \frac{S}{f_{rel}}$, then in any optimal solution of* TRI-CRIT-CHAIN*, all tasks that are neither re-executed nor replicated are executed at speed f_{rel}.*

Proof. The proof for $p = 1$ (re-execution) was given by Proposition 6.1. We prove the result for $p \geq 2$, which corresponds to the case with replication and no re-execution (see Lemma 7.2). Note first that since $D > \frac{S}{f_{\texttt{rel}}}$, if no task is replicated, we have enough time to execute all tasks at speed $f_{\texttt{rel}}$.

Now, let us consider that task T_i is replicated at speed f_i (recall that both replicas are executed at the same speed, see Lemma 7.3), and task T_j is executed only once at speed f_j. Then, we have $f_j \geq f_{\texttt{rel}}$ (reliability constraint on T_j), and $\frac{1}{\sqrt{2}} f_{\texttt{rel}} \geq f_i$ (otherwise, executing T_i only once at speed $f_{\texttt{rel}}$ would improve both the energy and the execution time while matching the reliability constraint).

If $f_j > f_{\texttt{rel}}$, let us show that we can rather execute T_j at speed $f_{\texttt{rel}}$ and T_i at a new speed $f_i' > f_i$, while keeping the same deadline: $\frac{w_i}{f_i'} + \frac{w_j}{f_{\texttt{rel}}} = \frac{w_i}{f_i} + \frac{w_j}{f_j}$. The energy consumption is then $2w_i f_i'^2 + w_j f_{\texttt{rel}}^2$. Moreover, we know that the minimum of the function $2w_i f_1^2 + w_j f_2^2$, given that $\frac{w_i}{f_1} + \frac{w_j}{f_2}$ is a constant (where f_1 and f_2 are the unknowns), is obtained for $f_1 = \frac{1}{2^{1/3}} f_2$, thanks to Theorem 5.1: the constraints are identical to a fork graph with $w_0 = w_j$ and $w_1 = w_2 = w_i$, and hence $f_1 = f_2 \times \frac{w_1}{(2w_1^3)^{\frac{1}{3}}}$. Therefore, if the optimal speed of T_j (that is f_2) is strictly greater than $f_{\texttt{rel}}$, then the optimal speed for T_i is $f_i' = f_1 = \frac{1}{2^{1/3}} f_2 > \frac{1}{2^{1/2}} f_2 > \frac{1}{2^{1/2}} f_{\texttt{rel}}$, that means that we can improve both energy and execution time by executing T_i only once at speed $f_{\texttt{rel}}$. Otherwise, the speed of T_j is further constrained by $f_{\texttt{rel}}$, hence the previous inequality ($f_1 = \frac{1}{2^{1/3}} f_2$) does not hold anymore, and the function is minimized for $f_2 = f_{\texttt{rel}}$. The value of f_i' can be easily deduced from the constraint on the deadline. This proves that all tasks that are not replicated are executed at speed $f_{\texttt{rel}}$.

■

Let V_r be the subset $V_r \subseteq V$ of tasks that are either re-executed or replicated, and let $X = \sum_{T_i \in V_r} w_i$. According to Proposition 7.1, the other tasks take a time $\frac{S-X}{f_{\texttt{rel}}}$, and the remaining time available for tasks of V_r is $D - \frac{S-X}{f_{\texttt{rel}}}$. Ideally, all tasks are executed at the same speed $f_{\texttt{re-ex}}$, as small as possible, so that the deadline constraint is met, as illustrated in Figure 7.1. We must also ensure that $f_{\texttt{re-ex}}$ is not smaller than f_{min}, and if this speed allows each task of V_r to meet the reliability constraint, then we can derive the energy of a schedule.

Following Proposition 7.1, we are able to precisely define $f_{\texttt{re-ex}}$, and give a closed form expression of the energy of a schedule when $f_{\texttt{re-ex}}$ is large enough.

Corollary 7.1. *Given a subset V_r of tasks re-executed or replicated, let $X = \sum_{T_i \in V_r} w_i$, and*

$$f_{re\text{-}ex} = \begin{cases} \max\left(f_{min}, \frac{2X}{Df_{rel}-S+X} f_{rel}\right) & \text{if } p = 1; \\ \max\left(f_{min}, \frac{X}{Df_{rel}-S+X} f_{rel}\right) & \text{if } p \geq 2. \end{cases}$$

Then, if $f_{re\text{-}ex} \geq \max_{T_i \in V_r} f_{\text{inf},i}$, all tasks of V_r are executed twice at speed $f_{re\text{-}ex}$, and the optimal energy consumption is

$$(S - X)f_{rel}^2 + 2Xf_{re\text{-}ex}^2. \tag{7.1}$$

Note that the energy consumption only depends on X, and therefore TRI-CRIT-CHAIN *is equivalent in this case to the problem of finding the optimal set of tasks that have to be re-executed or replicated.*

Proof. Given a deadline D, the problem is to find the set of re-executed (or replicated) tasks, and the speed of each task. Thanks to Proposition 7.1, we know that the tasks that are not in this set are executed at speed f_{rel}, and given the set of tasks re-executed or replicated, we can easily compute the optimal speed to execute each task in order to minimize the energy consumption. All tasks are executed at the same speed: the proof for $p = 1$ (re-execution) can be found in Proposition 6.1. We prove the result for $p \geq 2$, which corresponds to the case with replication and no re-execution (see Lemma 7.2). Suppose that T_i and T_j are executed twice at speeds $f_i > f_j \geq \max(f_{\text{inf},i}, f_{\text{inf},j})$, let $\tilde{f} = f_i f_j \frac{w_i + w_j}{w_i f_j + w_j f_i}$. Then $f_i > \tilde{f} > f_j$, and therefore, we can execute both tasks at speed $\tilde{f}$ while keeping the same deadline ($\frac{w_i}{\tilde{f}} + \frac{w_j}{\tilde{f}} = \frac{w_i}{f_i} + \frac{w_j}{f_j}$) and matching the reliability constraints (since $\tilde{f} \geq \max(f_{\text{inf},i}, f_{\text{inf},j})$, then two executions of task T_i or T_j at speed $\tilde{f}$ match the reliability constraint). By convexity, such an execution gives a smaller energy consumption. We can iterate on all the tasks that are replicated. Finally, if $f_{\text{re-ex}} \geq \max_{T_i \in V_r} f_{\text{inf},i}$ we have the result.

To conclude, we have $\lambda \frac{X}{f_{\text{re-ex}}} + \frac{S-X}{f_{\text{rel}}} = D$, with $\lambda = 1$ in the case of replication ($p \geq 2$), and $\lambda = 2$ in the case of re-execution ($p = 1$), hence the corollary. ■

Re-execution speeds. We are now ready to compute the optimal solution, given a subset $V_r \subseteq V$. We have not accounted yet for tasks $T_i \in V_r$ such that $f_{\text{inf},i} > f_{\text{re-ex}}$. In this case, T_i is executed at speed $f_{\text{inf},i}$, and all the other tasks are (tentatively) executed at a new speed $f_{\text{re-ex}}^{\text{new}} \leq f_{\text{re-ex}}$ such that D is exactly met. We do this iteratively until there are no more tasks T_i such that $f_{\text{inf},i} > f_{\text{re-ex}}^{\text{new}}$. Using the procedure COMPUTE_$V_l(V_r)$ (see Algorithm 4), we can therefore compute the optimal energy consumption in a time polynomial in $|V_r|$. We denote by V_l the set of tasks that are re-executed at speed $f_{\text{inf},i}$ (it is a subset of V_r, the set of tasks that are re-executed). Note that all tasks of $V_r \setminus V_l$ are re-executed at the speed $f_{\text{re-ex}}$ returned by COMPUTE_$V_l(V_r)$.

Let $(V_l, f_{\text{re-ex}})$ be the result of COMPUTE_$V_l(V_r)$. Then the optimal energy consumption is $(S - X)f_{\text{rel}}^2 + \sum_{T_i \in V_l} 2w_i f_{\text{inf},i}^2 + \sum_{T_i \in V_r \setminus V_l} 2w_i f_{\text{re-ex}}^2$.

Corollary 7.2. *If $D > \frac{S}{f_{rel}}$,* TRI-CRIT-CHAIN *can be solved using an exponential time exact algorithm.*

Proof. The algorithm computes for every subset V_r of tasks the energy consumption if all tasks in this subset are re-executed, and it chooses a subset with the minimal energy consumption, that corresponds to an optimal solution. It takes an exponential time to compute every subset $V_r \subseteq V$, with $|V| = n$. ■

Thanks to Corollary 7.1, we are also able to identify problem instances that can be solved in polynomial time.

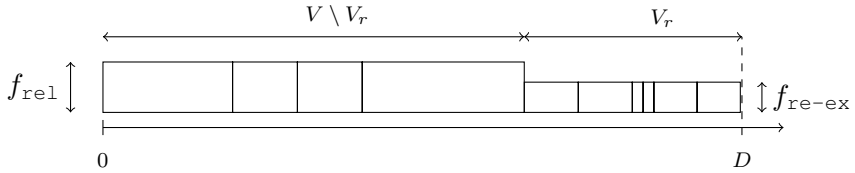


Figure 7.1: Illustration of the set V_r and $f_{\text{re-ex}}$.

Algorithm 4: Computing re-execution speeds; tasks in V_r are re-executed.

procedure COMPUTE_$V_l(V_r)$
begin
$V_l^{(0)} = \emptyset$;
$$f_{\texttt{re-ex}}^{(0)} = \begin{cases} \max\left(f_{\min}, \frac{2X}{Df_{\texttt{rel}}-S+X} f_{\texttt{rel}}\right) & \text{if } p = 1; \\ \max\left(f_{\min}, \frac{X}{Df_{\texttt{rel}}-S+X} f_{\texttt{rel}}\right) & \text{if } p \geq 2. \end{cases}$$
$j = 0$;
while $j = 0$ *or* $V_l^{(j)} \neq V_l^{(j-1)}$ **do**
$j := j + 1$;
$V_l^{(j)} = V_l^{(j-1)} \cup \{T_i \in V_r \mid f_{\text{inf},i} > f_{\texttt{re-ex}}^{(j-1)}\}$;
$$f_{\texttt{re-ex}}^{(j)} = \begin{cases} \max\left(f_{\min}, \frac{\sum_{T_i \in V_r \setminus V_l^{(j)}} 2w_i}{D - \frac{S-X}{f_{\texttt{rel}}} - \sum_{T_i \in V_l^{(j)}} \frac{2w_i}{f_{\text{inf},i}}}\right) & \text{if } p = 1; \\ \max\left(f_{\min}, \frac{\sum_{T_i \in V_r \setminus V_l^{(j)}} w_i}{D - \frac{S-X}{f_{\texttt{rel}}} - \sum_{T_i \in V_l^{(j)}} \frac{w_i}{f_{\text{inf},i}}}\right) & \text{if } p \geq 2. \end{cases}$$
return $(V_l^{(j)}, f_{\texttt{re-ex}}^{(j)})$;

Theorem 7.1. TRI-CRIT-CHAIN *can be solved in polynomial time in the following cases:*

1. $D \leq \frac{S}{f_{rel}}$ *(no re-execution nor replication);*
2. $p = 1$, $D \geq \frac{1+c}{c}\frac{S}{f_{rel}}$, *where* c *is the only positive solution to the polynomial* $7X^3+21X^2-3X-1=0$, *and hence* $c = 4\sqrt{\frac{2}{7}}\cos\frac{1}{3}(\pi - \tan^{-1}\frac{1}{\sqrt{7}}) - 1 \quad (c \approx 0.2838)$, *and for* $1 \leq i \leq n$, $f_{\text{inf},i} \leq \frac{2c}{1+c} f_{rel}$ *(all tasks can be re-executed);*
3. $p \geq 2$, $D \geq 2\frac{S}{f_{rel}}$, *and for* $1 \leq i \leq n$, $f_{\text{inf},i} \leq \frac{1}{2} f_{rel}$ *(all tasks can be replicated).*

Proof. First note that when $D \leq \frac{S}{f_{\texttt{rel}}}$, the optimal solution is to execute each task only once, at speed $\frac{S}{D}$, since $S/D \geq f_{\texttt{rel}}$. Indeed, this solution matches both reliability and makespan constraints, and it was proven to be the optimal solution in Proposition 5.2 (it is easy to see that replication or re-execution would only increase the energy consumption).

Let us now consider that $D > \frac{S}{f_{\texttt{rel}}}$. We aim at showing that the minimum of the energy function is reached when the total weight of the re-executed or replicated tasks is

$$X = \begin{cases} c(Df_{\texttt{rel}} - S) & \text{if } p = 1; \\ (Df_{\texttt{rel}} - S) & \text{if } p \geq 2. \end{cases} \tag{7.2}$$

Necessarily, when this total weight is greater than S, the optimal solution is to re-execute or replicate all the tasks, hence the theorem. We consider the two cases $p = 1$ and $p \geq 2$.

Case 1 ($p = 1$). We want to show that the minimum energy is reached when the total weight of the subset of tasks is exactly $c(Df_{\texttt{rel}} - S)$. Let $I = \{i \mid T_i \text{ is executed twice in the solution}\}$, and let $X = \sum_{i \in I} w_i$.

We saw in Corollary 7.1 that the energy consumption cannot be lower than $(S-X)f_{\texttt{rel}}^2+2Xf_{\texttt{re-ex}}^2$, where $f_{\texttt{re-ex}} = \frac{2X}{Df_{\texttt{rel}}-S+X}f_{\texttt{rel}}$. Therefore, we want to minimize

$$E(X) = (S-X)f_{\texttt{rel}}^2 + 2X\left(\frac{2X}{Df_{\texttt{rel}}-S+X}f_{\texttt{rel}}\right)^2.$$

If we differentiate E, we can see that the minimum is reached when

$$-1 + \frac{24X^2}{(Df_{\texttt{rel}}-S+X)^2} - \frac{16X^3}{(Df_{\texttt{rel}}-S+X)^3} = 0,$$

that is, $-(Df_{\texttt{rel}}-S+X)^3 + 24X^2(Df_{\texttt{rel}}-S+X) - 16X^3 = 0$, or

$$7X^3 + 21(Df_{\texttt{rel}}-S)X^2 - 3(Df_{\texttt{rel}}-S)^2X - (Df_{\texttt{rel}}-S)^3 = 0.$$

The only positive solution to this equation is

$$X = c(Df_{\texttt{rel}}-S),$$

and therefore the minimum is reached for this value of X, and then $f_{\texttt{re-ex}} = \frac{2c}{1+c}f_{\texttt{rel}}$.

When $X \geq S$, re-executing each task is the best strategy to minimize the energy consumption, and that corresponds to the case $D \geq \frac{1+c}{c}\frac{S}{f_{\texttt{rel}}}$. The re-execution speed may then be lower than $\frac{2c}{1+c}f_{\texttt{rel}}$. Therefore, it may happen that $f_{\text{inf},i} > f_{\texttt{re-ex}}$ for some task T_i. However, even with a tighter deadline, it would be better to re-execute T_i at speed $\frac{2c}{1+c}f_{\texttt{rel}}$ rather than to execute it only once at speed $f_{\texttt{rel}}$. Therefore, since $f_{\text{inf},i} \leq \frac{2c}{1+c}f_{\texttt{rel}}$, it is optimal to re-execute T_i, at the lowest possible speed, i.e., $f_{\text{inf},i}$. Note that this changes the value of $f_{\texttt{re-ex}}$, and the call to COMPUTE_$V_l(V)$ (see Algorithm 4) returns tasks that are executed at speed $f_{\text{inf},i}$, together with the re-execution speed for all the other tasks.

Case 2 ($p \geq 2$). Similarly, we want to show that, in this case, the minimum energy is reached when the total weight of the subset of tasks that are replicated is exactly $Df_{\texttt{rel}} - S$. Let $I = \{i \mid T_i \text{ is executed twice in the solution}\}$, and let $X = \sum_{i\in I} w_i$.

We saw in Corollary 7.1 that the energy consumption cannot be lower than $(S-X)f_{\texttt{rel}}^2+2Xf_{\texttt{re-ex}}^2$ where $f_{\texttt{re-ex}} = \frac{X}{Df_{\texttt{rel}}-S+X}f_{\texttt{rel}}$. Therefore, we want to minimize

$$E(X) = (S-X)f_{\texttt{rel}}^2 + 2X\left(\frac{X}{Df_{\texttt{rel}}-S+X}f_{\texttt{rel}}\right)^2.$$

If we differentiate E, we can see that the minimum is reached when

$$-1 + \frac{6X^2}{(Df_{\texttt{rel}}-S+X)^2} - \frac{4X^3}{(Df_{\texttt{rel}}-S+X)^3} = 0,$$

that is, $-(Df_{\texttt{rel}}-S+X)^3 + 6X^2(Df_{\texttt{rel}}-S+X) - 4X^3 = 0$, or

$$X^3 + 3(Df_{\texttt{rel}}-S)X^2 - 3(Df_{\texttt{rel}}-S)^2X - (Df_{\texttt{rel}}-S)^3 = 0.$$

The only positive solution to this equation is

$$X = Df_{\texttt{rel}}-S,$$

and therefore the minimum is reached for this value of X, and then $f_{\texttt{re-ex}} = \frac{1}{2}f_{\texttt{rel}}$.

When $X \geq S$, replicating each task is the best strategy to minimize the energy consumption, and that corresponds to the case $D \geq \frac{2S}{f_{\texttt{rel}}}$. Similarly to Case 1, it is easy to see that each task should be replicated, even if $f_{\text{inf},i} > f_{\texttt{re-ex}}$, since $f_{\text{inf},i} \leq \frac{1}{2}f_{\texttt{rel}}$. The optimal solution can also be obtained with a call to COMPUTE_$V_l(V)$. ■

7.3.2 FPTAS for TRI-CRIT-CHAIN

We derive in this section a fully polynomial-time approximation scheme (FPTAS) for TRI-CRIT-CHAIN, based on the FPTAS for SUBSET-SUM [34], and the results of Section 7.3.1. Without loss of generality, we use the term *replication* for either re-execution or replication, since both scenarios have already been clearly identified. The problem consists in identifying the set of replicated tasks V_r, and then the optimal solution can be derived from Corollary 7.1; it depends only on the total weight of these tasks, $\sum_{T_i \in V_r} w_i$, denoted in the following as $w(V_r)$.

Note that we do not account in this section for $f_{\text{inf},i}$ or f_{min} for readability reasons: $f_{\text{inf},i}$ can usually be neglected because $\lambda_0 w_i / f$ is supposed to be very small whatever f, and f_{min} simply adds subcases to the proofs (rather than an execution at speed f, the speed should be $\max(f, f_{\text{min}})$).

First we introduce a few preliminary functions in Algorithm 5, and we exhibit their properties. These are the basis of the approximation algorithm.

When $D > \frac{S}{f_{\text{rel}}}$, X-OPT$(V, D, p)$ returns the optimal value for the weight $w(V_r)$ of the subset of replicated tasks V_r, i.e., the value that minimizes the energy consumption for TRI-CRIT-CHAIN, according to Equation (7.2). The optimality comes directly from the proof of Theorem 7.1.

Given a value X, which corresponds to $w(V_r)$, ENERGY (V, D, p, X) returns the optimal energy consumption when a subset of tasks V_r is replicated.

Then, the function TRIM(L, ε, X) trims a sorted list of numbers $L = [L_0, \ldots, L_{m-1}]$ in time $O(m)$, given L and ε. L is sorted into non decreasing order. The function returns a trimmed list, where two consecutive elements differ by at least a factor $(1 + \varepsilon)$, except the last element, that is the smallest element of L strictly greater than X. This trimming procedure is quite similar to that used for SUBSET-SUM [34], except that the latter keeps only elements lower than X. Indeed, SUBSET-SUM can be expressed as follows: given n strictly positive integers $a_1, \ldots, a_n$, and a positive integer X, we wish to find a subset I of $\{1, \ldots, n\}$ such that $\sum_{i \in I} w_i$ is as large as possible, but not larger than X. In our case, the optimal solution may be obtained either by approaching X by below or by above.

Given a list $L = [L_0, \ldots, L_{m-1}]$, ADD-LIST$(L, x)$ adds element x at the end of list L (i.e., it returns the list $[L_0, \ldots, L_{m-1}, x]$); $L + w$ is the list $[L_0 + w, \ldots, L_{m-1} + w]$; and MERGE-LISTS$(L, L')$ merges two sorted lists (and returns a sorted list).

Finally, the approximation algorithm is APPROX-CHAIN(V, D, p, ε) (see Algorithm 5), where $0 < \varepsilon < 1$, and it returns an energy consumption E that is not greater than $(1 + \varepsilon)$ times the optimal energy consumption.

We now prove that this approximation scheme is an FPTAS:

Theorem 7.2. APPROX-CHAIN *is a fully polynomial-time approximation scheme for* TRI-CRIT-CHAIN.

Proof. We assume that

- if $p = 1$, then $\frac{S}{f_{\text{rel}}} < D < \frac{1+c}{c} \frac{S}{f_{\text{rel}}} < 5 \frac{S}{f_{\text{rel}}}$;
- if $p \geq 2$, then $\frac{S}{f_{\text{rel}}} < D < 2 \frac{S}{f_{\text{rel}}}$;

otherwise the optimal solution is obtained in polynomial time (see Theorem 7.1).

Let $I_{\text{inf}} = \{V' \subseteq V \mid w(V') \leq \text{X-OPT}(V, D, p)\}$, and $I_{\text{sup}} = \{V'' \subseteq V \mid w(V'') > \text{X-OPT}(V, D, p)\}$. Note that I_{inf} is not empty, since $\emptyset \in I_{\text{inf}}$.

First we characterize the solution with the following lemma:

Lemma 7.4. *Suppose $D > \frac{S}{f_{\text{rel}}}$. Then in the solution of* TRI-CRIT-CHAIN*, the subset of replicated tasks V_r is either an element $V' \in I_{\text{inf}}$ such that $w(V')$ is maximum, or an element $V'' \in I_{\text{sup}}$ such that $w(V'')$ is minimum.*

Algorithm 5: Approximation algorithm for TRI-CRIT-CHAIN.

function X-OPT(V, D, p)
begin
  $S = \sum_{T_i \in V} w_i$;
  if $p = 1$ **then return** $c(Df_{rel} - S)$;
  else return $Df_{rel} - S$;

function ENERGY(V, D, p, X)
begin
  $S = \sum_{T_i \in V} w_i$;
  if $p{=}1$ **then return** $(S{-}X)f_{rel}^2{+}2X\left(\max\left(f_{min}, \frac{2X}{Df_{rel}-S+X} f_{rel}\right)\right)^2$;
  else return $(S - X)f_{rel}^2 + 2X\left(\max\left(f_{min}, \frac{X}{Df_{rel}-S+X} f_{rel}\right)\right)^2$;

function TRIM(L, ε, X)
begin
  $m = |L|$; $L = [L_0, \ldots, L_{m-1}]$; $L' = [L_0]$; $last = L_0$;
  for $i = 1$ **to** $m - 1$ **do**
    if *($last \leq X$ and $L_i > X$) or* $L_i > last \times (1 + \varepsilon)$ **then**
      $L' =$ ADD-LIST(L', L_i); $last = L_i$;
  return L';

function APPROX-CHAIN(V, D, p, ε)
begin
  $X = \lfloor$X-OPT$(V, D, p)\rfloor$; $n = |V|$; $L^{(0)} = [0]$;
  for $i = 1$ **to** n **do**
    $L^{(i)} =$ MERGE-LISTS$(L^{(i-1)}, L^{(i-1)} + w_i)$;
    $L^{(i)} =$ TRIM$(L^{(i)}, \varepsilon/(28 \times 2n), X)$;
  Let $Y_1 \leq Y_2$ be the two largest elements of $L^{(n)}$;
  return $\min($ENERGY$(V, D, p, Y_1),$ ENERGY$(V, D, p, Y_2))$;

Proof. Recall first that according to Proposition 7.1, the energy consumption of a linear chain is not dependent on the number of tasks replicated, but only on the sum of their weights.

Then the lemma is obvious by convexity of the functions, and because X-OPT returns the optimal value of $w(V_r)$, the weight of the replicated tasks. Therefore, the closest the weight of the set of replicated tasks is to the optimal weight, the better the solution is.

■

We are now ready to prove Theorem 7.2. Let $X_1 = \max_{V_1 \in I_{\text{inf}}} w(V_1)$, and $X_2 = \min_{V_2 \in I_{\text{sup}}} w(V_2)$. Thanks to Lemma 7.4, the optimal set of replicated tasks V_o is such that $X_o = w(V_o) = X_1$ or $X_o = X_2$. The corresponding energy consumption is (Corollary 7.1):

$$E_{opt} = \begin{cases} (S - X_o) f_{\text{rel}}^2 + \frac{(2X_o)^3}{(Df_{\text{rel}} - S + X_o)^2} f_{\text{rel}}^2 & \text{if } p = 1; \\ (S - X_o) f_{\text{rel}}^2 + \frac{2X_o^3}{(Df_{\text{rel}} - S + X_o)^2} f_{\text{rel}}^2 & \text{if } p \geq 2. \end{cases}$$

The solution returned by APPROX-CHAIN corresponds either to Y_1 or to Y_2, where Y_1 and Y_2 are the two largest elements of the trimmed list. We first prove that at least one of these two elements, denoted X_a, is such that $X_a \leq X_o \leq (1 + \varepsilon')X_a$, where $\varepsilon' = \frac{\varepsilon}{28}$.

Existence of X_a such that $X_a \leq X_o \leq (1 + \varepsilon')X_a$.

(a) If $Y_2 > X$, then Y_1 is the value obtained by the FPTAS for SUBSET-SUM [34] with the approximation ratio ε', since it is the largest value not greater than X, and our algorithm is identical for such values. Moreover, note that X_1 is the optimal solution of SUBSET-SUM by definition, and therefore $Y_1 \leq X_1 < (1 + \varepsilon')Y_1$. If $X_o = X_1$, the value $X_a = Y_1$ satisfies the property.

If $X_o = X_2$, we prove that the property remains valid, by considering the SUBSET-SUM problem with a bound X_2 instead of X. Then, since $Y_2 > X$, we have $Y_2 \geq X_2$ by definition of X_2. Moreover, APPROX-CHAIN is not removing any element of the list greater than Y_2, and therefore all elements between X and X_2 are kept, similarly to the FPTAS for SUBSET-SUM. If $Y_2 = X_2$, then $X_a = Y_2$ satisfies the property. Otherwise, Y_1 is the result of the FPTAS for SUBSET-SUM with a bound X_2, whose optimal solution is X_2, and therefore Y_1 is such that $Y_1 \leq X_2 < (1 + \varepsilon')Y_1$; $X_a = Y_1$ satisfies the property.

(b) If $Y_2 \leq X$, no elements greater than X have been removed from the lists, and APPROX-CHAIN has been identical to the FPTAS for SUBSET-SUM. Then, $X_a = Y_2$ is the solution, that is valid both for SUBSET-SUM applied with the original bound X (optimal solution X_1), and with the modified bound X_2 (optimal solution X_2). Therefore, $Y_2 \leq X_1 < (1 + \varepsilon')Y_2$ and $Y_2 \leq X_2 < (1 + \varepsilon')Y_2$, which concludes the proof.

We have shown that there always is X_a (either Y_1 or Y_2) such that $X_a \leq X_o < (1 + \varepsilon')X_a$. Next, we show that the energy E_a obtained with this value X_a is such that $E_{opt} \leq E_a \leq (1 + \varepsilon)E_{opt}$.

Approximation ratio on the energy: $E_a \leq (1 + \varepsilon)E_{opt}$. Let us consider first that $p \geq 2$. Then we have

$$E_a = (S - X_a) f_{\text{rel}}^2 + \frac{2X_a^3}{(Df_{\text{rel}} - S + X_a)^2} f_{\text{rel}}^2.$$

Re-using the previous inequalities on X_a, we obtain:

$$\frac{E_a}{f_{\text{rel}}^2} \leq S - \frac{X_o}{1 + \varepsilon'} + \frac{2X_o^3}{(Df_{\text{rel}} - S + \frac{X_o}{1+\varepsilon'})^2}.$$

Then, this can be rewritten so that E_{opt} appears:

$$\frac{E_a}{f_{\text{rel}}^2} \leq \left(\frac{1}{1+\varepsilon'}(S - X_o) + \frac{\varepsilon'}{1+\varepsilon'}S\right) + \left((1+\varepsilon')^2 \frac{2X_o^3}{((1+\varepsilon')(Df_{\text{rel}} - S) + X_o)^2}\right)$$

$$\begin{aligned}
\frac{E_a}{f_{\text{rel}}^2} &\leq \big((S - X_o) + \varepsilon' S\big) \\
&\quad + \left((1+\varepsilon')^2 \frac{2X_o^3}{(Df_{\text{rel}} - S + X_o)^2}\right) \\
&\leq \big((S - X_o) + \varepsilon' S\big) \\
&\quad + \left((1+\varepsilon')^2 (\frac{E_{opt}}{f_{\text{rel}}^2} - (S - X_o))\right) \\
&\leq (1+\varepsilon')^2 \frac{E_{opt}}{f_{\text{rel}}^2} \\
&\quad - ((1+\varepsilon')^2 - 1)(S - X_o) + \varepsilon' S \\
&\leq (1+\varepsilon')^2 \frac{E_{opt}}{f_{\text{rel}}^2} + \varepsilon' S.
\end{aligned}$$

The case $p = 1$ leads to the same inequality; the only difference is in the energy E_a, where $2X_a^3$ is replaced by $(2X_a)^3$, and the same difference holds for E_{opt} ($2X_o^3$ is replaced by $(2X_o)^3$).

Finally, note that with no reliability constraints, each task is executed only once at speed S/D, and therefore the energy consumption is at least $E_{opt} \geq S\frac{S^2}{D^2}$. Moreover, by hypothesis, $D < \frac{5S}{f_{\text{rel}}}$ (for $p \geq 1$). Therefore, $S < \frac{25E_{opt}}{f_{\text{rel}}^2}$ and $\frac{E_a}{f_{\text{rel}}^2} < (1+\varepsilon')^2 \frac{E_{opt}}{f_{\text{rel}}^2} + \varepsilon' \frac{25E_{opt}}{f_{\text{rel}}^2}$.

We conclude that

$$\frac{E_a}{E_{opt}} < 1 + 27\varepsilon' + \varepsilon'^2 < 1 + 28\varepsilon' = 1 + \varepsilon.$$

Conclusion. The energy consumption returned by APPROX-CHAIN, denoted as E_{algo}, is such that $E_{algo} \leq E_a$, since we take the minimum out of the consumption obtained for Y_1 or Y_2, and X_a is either Y_1 or Y_2. Therefore,

$$E_{algo} \leq (1+\varepsilon)E_{opt}.$$

It is clear that the algorithm is polynomial both in the size of the instance and in $\frac{1}{\varepsilon}$, given that the trimming function and APPROX-CHAIN have the same complexity as in the original approximation scheme for SUBSET-SUM (see Cormen et al. [34]), and all other operations are polynomial in the problem size (X-OPT, ENERGY). ■

7.4 Independent tasks

In this section, we focus on the problem of scheduling independent tasks, TRI-CRIT-INDEP. Similarly to TRI-CRIT-CHAIN, we know that TRI-CRIT-INDEP is NP-hard, even on a single processor. We first prove in Section 7.4.1 that there exists no constant factor approximation algorithm for this problem, unless P=NP. We discuss and characterize solutions to TRI-CRIT-INDEP in Section 7.4.2, while

highlighting the intrinsic difficulty of the problem. The core result is a constant factor approximation algorithm with a relaxation on the constraint on the makespan (Section 7.4.3).

It is more difficult to characterize the feasibility of the problem with independent tasks when $p \geq 2$ than for TRI-CRIT-CHAIN. Indeed, deciding whether there is a solution or not is NP-hard (trivial reduction from 2-PARTITION with $p = 2$ and a tight deadline: $S/2f_{\text{max}} = D$).

7.4.1 Inapproximability of TRI-CRIT-INDEP

For TRI-CRIT-INDEP, a λ-approximation algorithm is a polynomial-time algorithm that returns a solution of energy consumption $E_{algo} \leq \lambda E_{opt}$, where E_{opt} is the energy consumption of the optimal solution, if there is a solution to the problem. Because the feasibility problem is NP-hard, we prove that there is no λ-approximation algorithm, unless P=NP, because such an algorithm would allow us to decide on the feasibility of the problem, and hence to solve in polynomial time an NP-complete problem.

Lemma 7.5. *For all $\lambda > 1$, there does not exist any λ-approximation algorithm for* TRI-CRIT-INDEP*, unless P=NP.*

Proof. By contradiction, let us assume that there is a λ-approximation algorithm for TRI-CRIT-INDEP. We consider an instance $\mathcal{I}_1$ of 2-PARTITION: given n strictly positive integers $a_1, \ldots, a_n$, does there exist a subset I of $\{1,\ldots,n\}$ such that $\sum_{i\in I} a_i = \sum_{i\notin I} a_i$? Let $S = \sum_{i=1}^{n} a_i$.

We build the following instance $\mathcal{I}_2$ of our problem. We have n independent tasks T_i to be mapped on $p = 2$ processors, and:

- task T_i has a weight $w_i = a_i$;
- $f_{\text{min}} = f_{\texttt{rel}} = f_{\text{max}} = S/2$;
- $D = 1$.

We use the λ-approximation algorithm to solve $\mathcal{I}_2$, and the solution of the algorithm E_{algo} is such that $E_{algo} \leq \lambda E_{opt}$, where E_{opt} is the optimal solution. We consider the two following cases.
(i) If the λ-approximation algorithm returns a solution, then necessarily all tasks are executed exactly once at speed f_{max}, since $\sum_{i=1}^{n} w_i/f_{\text{max}} = 2$ and there are two processors. Moreover, because of the makespan constraint, the load on each processor is equal. Let I be the indices of the tasks executed on the first processor. We have $\sum_{i\in I} a_i = \sum_{i\notin I} a_i$, and therefore I is also a solution to $\mathcal{I}_1$.
(ii) If the λ-approximation algorithm does not return a solution, then there is no solution to $\mathcal{I}_1$. Otherwise, if I is a solution to $\mathcal{I}_1$, there is a solution to $\mathcal{I}_2$ such that tasks of I are executed on the first processor, and the other tasks are executed on the second processor. Since $E_{algo} \leq \lambda E_{opt}$, the approximation algorithm should have returned a valid solution.

Therefore, the result of the algorithm for $\mathcal{I}_2$ allows us to conclude in polynomial time whether there is a solution to the instance $\mathcal{I}_1$ of 2-PARTITION or not. Since 2-PARTITION is NP-complete [48], the inapproximability result is true unless P=NP. ■

7.4.2 Characterization

As discussed in Section 7.1, the problem of scheduling independent tasks is usually close to a problem of load balancing, and can be efficiently approximated for various mono-criterion versions of the problem (minimizing the makespan or the energy, for instance). However, the tri-criteria problem turns out to be much harder, and cannot be approximated, as seen in Section 7.4.1, even when reliability is not a constraint.

Adding reliability further complicates the problem, since we no longer have the property that on each processor, there is a constant execution speed for the tasks executed on this processor. Indeed,

some processors may process both tasks that are not replicated (or re-executed), hence at speed $f_{\texttt{rel}}$, and replicated tasks at a slower speed. Similarly to Section 7.3.2, we use the term *replication* for either re-execution or replication; if a task is replicated, it means it is executed two times, and it appears two times in the load of processors, be it the same processor or two distinct processors.

Furthermore, contrary to the TRI-CRIT-CHAIN problem, we do not always have the same execution speed for both executions of a task, as in Lemma 7.3:

Proposition 7.2. *In an optimal solution of* TRI-CRIT-INDEP*, if a task T_i is executed twice:*

- *if both executions are on the same processor, then both are executed at the same speed that is at most $\frac{1}{\sqrt{2}} f_{rel}$;*
- *however, when the two executions of this task are on distinct processors, then they are not necessarily executed at the same speed. Furthermore, one of the two speeds can be greater than $\frac{1}{\sqrt{2}} f_{rel}$.*

Moreover, we have $w_i < \frac{1}{\sqrt{2}} D f_{rel}$.

Proof. We start by proving the properties on the speeds. When both executions occur on the same processor, we showed in Lemma 6.2 the property: a single execution at speed $f_{\texttt{rel}}$ leads to a better energy consumption (and a lower execution time) if both executions are executed at a speed greater than $\frac{1}{\sqrt{2}} f_{\texttt{rel}}$.

In the case of distinct processors, we give below an example in which the optimal solution uses different speeds for a replicated task, with one speed greater than $\frac{1}{\sqrt{2}} f_{\texttt{rel}}$. Note that one of the speeds is necessarily at most $\frac{1}{\sqrt{2}} f_{\texttt{rel}}$, otherwise a solution with only one execution of this task at speed $f_{\texttt{rel}}$ would be better, similarly to the case with re-execution.

Consider a problem instance with two processors, $f_{\texttt{rel}} = f_{\max}$, $D = \frac{6.4}{f_{\max}}$, and three tasks such that $w_1 = 5$, $w_2 = 3$, and $w_3 = 1$. Because of the time constraints, T_1 and T_2 are necessarily executed on two distinct processors, and neither of them can be re-executed on its processor. The problem consists in scheduling task T_3 to minimize the energy consumption. There are three possibilities:

- T_3 is executed only once on any of the processors, at speed $f_{\texttt{rel}} = f_{\max}$;
- T_3 is executed twice on the same processor; it is executed on the same processor as T_2, hence having an execution time of $D - \frac{w_2}{f_{\max}} = \frac{3.4}{f_{\max}}$, and therefore both executions are done at a speed $\frac{2}{3.4} f_{\max}$;
- T_3 is executed once on the same processor as T_1 at a speed $\frac{1}{1.4} f_{\max}$, and once on the other processor at a speed $\frac{1}{3.4} f_{\max}$.

It is easy to see that the minimum energy consumption is obtained with the last solution, and that $\frac{1}{1.4} f_{\max} > \frac{1}{\sqrt{2}} f_{\texttt{rel}}$, hence the result.

Finally, note that since at least one of the executions of the task should be at a speed lower than $\frac{1}{\sqrt{2}} f_{\texttt{rel}}$, and since the deadline is D, in order to match the deadline, the weight of the replicated task has to be strictly lower than $\frac{1}{\sqrt{2}} D f_{\texttt{rel}}$. ■

Because of this proposition, usual load balancing algorithms are likely to fail, since processors handling only non-replicated tasks should have a much higher load, and speeds of replicated tasks may be very different from one processor to another in the optimal solution.

We now derive lower bounds on the energy consumption, that will be useful to design an approximation algorithm in the next section.

Proposition 7.3 (Lower bound without reliability)**.** *The optimal solution of* TRI-CRIT-INDEP *cannot have an energy lower than $\frac{S^3}{(pD)^2}$.*

Proof. Let us consider the problem of minimizing the energy consumption, with a deadline constraint D, but without accounting for the constraint on reliability. A lower bound is obtained if the load on each processor is exactly equal to $\frac{S}{p}$, and the speed of each processor is constant and equal to $\frac{S}{pD}$. The corresponding energy consumption is $S \times \left(\frac{S}{pD}\right)^2$, hence the bound. ■

However, if the speed $\frac{S}{pD}$ is small compared to $f_{\texttt{rel}}$, the bound is very optimistic since reliability constraints are not matched at all. Indeed, replication must be used in such a case. We investigate bounds that account for replication in the following, using the optimal solution of the TRI-CRIT-CHAIN problem.

Proposition 7.4 (Lower bound using linear chains)**.** *For the* TRI-CRIT-INDEP *problem, the optimal solution cannot have an energy lower than the optimal solution to the* TRI-CRIT-CHAIN *problem on a single processor with a deadline pD, where the weight of the re-executed tasks is lower than $\frac{1}{\sqrt{2}} D f_{\texttt{rel}}$.*

Proof. We can transform any solution to the TRI-CRIT-INDEP problem into a solution to the TRI-CRIT-CHAIN problem with deadline pD and a single processor. Tasks are arbitrarily ordered as a linear chain, and the solution uses the same number of executions and the same speed(s) for each task. It is easy to see that the TRI-CRIT-INDEP problem is more constrained, since the deadline on each processor must be enforced. The constraint on the weights of the re-executed tasks comes from Proposition 7.2. Therefore, the solution to the TRI-CRIT-CHAIN problem is a lower bound for TRI-CRIT-INDEP. ■

The optimal solution may however be far from this bound, since we do not know if the tasks that are re-executed on a chain with a long deadline pD can be executed at the same speed when the deadline is D. The constraint on the weight of the re-executed tasks allows us to improve slightly the bound, and this lower bound is the basis of the approximation algorithm that we design for TRI-CRIT-INDEP.

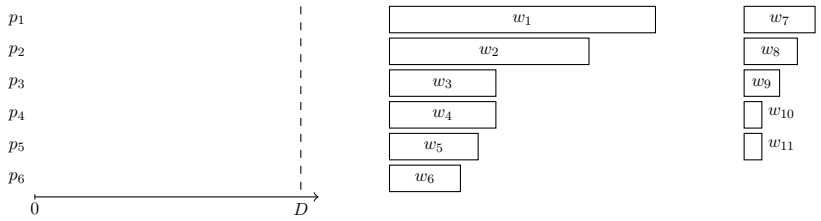
7.4.3 Approximation algorithm for TRI-CRIT-INDEP

We have seen in Section 7.4.1 that there exists no constant factor approximation algorithm for TRI-CRIT-INDEP, unless P=NP, even without accounting for the reliability constraint. This is due to the constraint on the makespan and the maximum speed $f_{\max}$. Therefore, in order to provide a constant factor approximation algorithm, we relax the constraint on the makespan and propose an (α, β)-approximation algorithm. The solution E_{algo} is such that $E_{algo} \leq \alpha \times E_{opt}$, where E_{opt} is the optimal solution with the deadline constraint D, and the makespan of the solution returned by the algorithm, M_{algo}, is such that $M_{algo} \leq \beta \times D$.

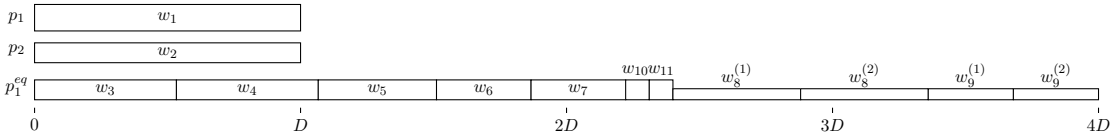
If the original problem with deadline D has no solution, because of the deadline relaxation, the (α, β)-approximation algorithm may or may not return a solution (contrarily to λ-approximation algorithms as in the proof of Lemma 7.5), but then there is no guarantee to ensure because there is no optimal solution. Therefore, we do not consider such cases for proving the correctness and guarantee of the algorithm. In particular, we assume that for all i, $w_i/f_{\max} \leq D$, and that $S/pf_{\max} \leq D$, otherwise we know that there is no solution.

The result of Section 7.4.1 means that for all $\alpha > 1$, there is no $(\alpha, 1)$-approximation algorithm for TRI-CRIT-INDEP, unless P=NP. Therefore, we present an algorithm that realizes a $(1 + \frac{1}{\beta^2}, \beta)$-approximation, where β can be slightly smaller than 2 and can take any arbitrarily large value: $\beta \geq \max\left(2 - \frac{3}{2p+1}, 2 - \frac{p+2}{4p+2}\right)$.

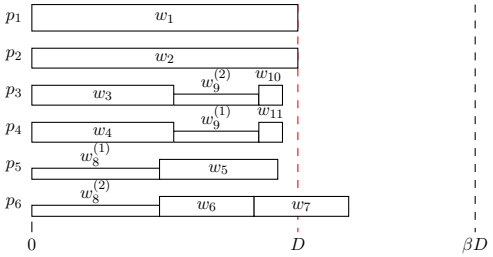
Algorithm. In the first step of the algorithm, we schedule each task with a big weight alone on one processor, with no replication. A task T_i is considered as *big* if $w_i \geq \max(\frac{S}{p}, D f_{\texttt{rel}})$. This step is



(a) Input: six processors and eleven tasks



(b) Schedule the *big* tasks on p_1 and p_2, and call APPROX-CHAIN with deadline $(6-2)D$ on the remaining tasks



(c) Greedy algorithm to schedule the new tasks

Figure 7.2: $\left(1+\frac{1}{\beta^2},\beta\right)$-approximation algorithm for independent tasks.

done in polynomial time: we sort the tasks by non-increasing weights, and then we check whether the current task is such that $w_i \geq \max(\frac{S}{p}, Df_{\texttt{rel}})$. If it is the case, we schedule the task alone on an unused processor and we let $S = S - w_i$ and $p = p - 1$. The procedure ends when the current task is small enough, i.e., all remaining tasks are such that $w_i < \max(\frac{S}{p}, Df_{\texttt{rel}})$, with the updated values of S and p. Note that there are always enough unused processors because selected big tasks are such that $w_i \geq \frac{S}{p}$, and therefore there cannot be more than p such tasks (and this is true at each step). When $p = 1$, either there is only one remaining task of size S, or there are only small tasks left.

These big tasks can be safely ignored in the remainder of the algorithm, hence the abuse of notations S and p for the remaining load and the remaining processors. Indeed, we will prove that this first step of the algorithm takes decisions that are identical to the optimal solution, and therefore these tasks that are executed once, alone on their processor, have the same energy consumption and the same deadline as in the optimal solution. The next step depends on the remaining load S:

- If $S > pDf_{\texttt{rel}}$, i.e., the remaining load is *large enough*, we do not use replication, but we schedule the tasks at speed $\frac{S}{pD}$, using a simple scheduling heuristic, LONGEST-PROCESSING-TIME [55]. Tasks are numbered by non increasing weights, and at each time step, we schedule the current task on the least loaded processor. Thanks to the lower bound of Proposition 7.3, the energy consumption is not greater than the optimal energy consumption, and we determine β such that the deadline is enforced.
- If $S \leq pDf_{\texttt{rel}}$, the previous bound is not good enough, and therefore we use the FPTAS on a

linear chain of tasks with deadline pD for TRI-CRIT-CHAIN (see Theorem 7.2). The FPTAS is called with

$$\varepsilon = \min\left(\frac{2w_{min}}{3S}\left(\frac{f_{\min}}{f_{\texttt{rel}}}\right)^2, \frac{1}{3\beta^2}\right), \tag{7.3}$$

where $w_{min} = \min_{1\leq i\leq n} w_i$. Note that it is slightly modified so that only tasks of weight $w < \frac{1}{\sqrt{2}} D f_{\texttt{rel}}$ can be replicated, and that we enforce a minimum speed $f_{\min}$. The FPTAS therefore determines which tasks should be executed twice, and it fixes all execution speeds.
We then use LONGEST-PROCESSING-TIME in order to map the tasks onto the p processors, at the speeds determined earlier. The new set of tasks includes both executions in case of replication, and tasks are sorted by non increasing execution times (since all speeds are fixed). At each time step, we schedule the current task on the least loaded processor. If some tasks cannot fit in one processor within the deadline βD, we re-execute them at speed $\frac{w_i}{\beta D}$ on two processors. Thanks to the lower bound of Proposition 7.4, we can bound the energy consumption in this case.

We illustrate the algorithm on an example in Figure 7.2, where eleven tasks must be mapped on six processors. For each task, we represent its execution speed as its height, and its execution time as its width. There are two *big* tasks, of weights w_1 and w_2, that are each mapped on a distinct processor. Then, we have $p = 4$ and we call APPROX-CHAIN with deadline $4D$; tasks T_8 and T_9 are replicated. Finally, LONGEST-PROCESSING-TIME greedily maps all instances of the tasks, slightly exceeding the original bound D, but all tasks fit within the extended deadline.

This algorithm leads to the following theorem:

Theorem 7.3. *For the problem* TRI-CRIT-INDEP, *there are* $\left(1+\frac{1}{\beta^2}, \beta\right)$*-approximation algorithms, for all* $\beta \geq \max\left(2-\frac{3}{2p+1}, 2-\frac{p+2}{4p+2}\right)$*, that run in polynomial time.*

Before proving Theorem 7.3, we give some preliminary results: we prove below the optimality of the first step of the algorithm, i.e., the optimal solution would schedule tasks of weight greater than $\max(\frac{S}{p}, D f_{\texttt{rel}})$ alone on a processor:

Proposition 7.5. *In any optimal solution to* TRI-CRIT-INDEP, *each task* T_i *such that* $w_i \geq \max(\frac{S}{p}, D f_{\texttt{rel}})$ *is executed only once, and it is alone on its processor.*

Proof. Let us prove the result by contradiction. Suppose that there exists a task T_1^* such that $w_1^* \geq \max(\frac{S}{p}, D f_{\texttt{rel}})$, and that this task is executed on processor p_1. Suppose also that there is another task T_2^* executed on p_1, with $w_2^* \leq w_1^*$, in an optimal solution. Necessarily, there exists a processor, say p_2, whose load is smaller than $\frac{S}{p}$, since the load of p_1 is greater than $\frac{S}{p}$. Let $w_1, \ldots, w_k$ be the weights of the tasks already scheduled on p_2, at speeds $f_1, \ldots, f_k$. We have $S_k = \sum_{i=1}^k w_i < \frac{S}{p} \leq w_1^*$. Let $f_* = \frac{w_1^*+w_2^*}{D}$ be the speed at which processor p_1 is executing tasks T_1^* and T_2^* (because the load of processor p_1 is greater than $D f_{\texttt{rel}}$, then with an argument similar to the one used in Theorem 7.1, all tasks should be executed at the same speed and the deadline is tight).

• If $S_k + w_2^* \geq D f_{\texttt{rel}}$, then a lower bound to the optimal solution is $E(opt) \geq (w_1^*+w_2^*) f_*^2 + S_k^3/D^2$, and $D^2 E(opt) \geq (w_1^* + w_2^*)^3 + S_k^3$ (this is the lower bound from Proposition 7.3 when we consider each processor independently). A new solution would be to execute T_2^* on p_2, obtaining an energy E such that $D^2 E = (w_1^*)^3 + (S_k + w_2^*)^3$ (the load of each processor is greater than $D f_{\texttt{rel}}$), and finally $E < E(opt)$ because $S_k < w_1^*$, and hence the contradiction.

• If $S_k + w_2^* < D f_{\texttt{rel}}$, all tasks on p_2 can be executed at a speed lower than $\frac{w_1^*}{D}$ (since $w_1 \geq D f_{\texttt{rel}}$), even when T_2^* is executed on p_2. On the one hand, we increase the speed of some tasks $w_1, \ldots, w_k$ that

were lower than $f_{\texttt{rel}}$ in order to gain a time w_2^*/f_*, that is the time required to fit task T_2^* on p_2, while running at the same speed as in the optimal solution. On the other hand, we decrease the speed of task T_1^* to use the time w_2^*/f_* that is now available.

We now prove that if tasks T_a and T_b are executed at speeds $f_a > f_b$, then it is strictly better to decrease f_a to $g_a = f_a - \epsilon$ (with $\epsilon > 0$), and increase f_b to g_b, while keeping the same total execution time, as long as $g_a \geq g_b$. The constraint on execution time writes

$$\frac{w_a}{f_a} + \frac{w_b}{f_b} = t = \frac{w_a}{f_a - \epsilon} + \frac{w_b}{g_b},$$

and therefore $g_b = \frac{w_b f_a - w_b \epsilon}{t f_a - w_a - t\epsilon}$. The difference of energy between the two solutions can be expressed as a function of ϵ:

$$h(\epsilon) = w_a f_a^2 + w_b f_b^2 - w_a (f_a - \epsilon)^2 - w_b g_b^2,$$

and we have $h(0) = 0$. The derivative is

$$h'(\epsilon) = 2w_a(f_a - \epsilon)\left(1 - \frac{w_b^3}{(tf_a - w_a - t\epsilon)^3}\right).$$

$h(\epsilon)$ is increasing when $h'(\epsilon) \geq 0$, that is as long as $w_b \leq tf_a - w_a - t\epsilon$, i.e., $f_a - \epsilon \geq \frac{w_a + w_b}{t}$. This corresponds to the case where $f_a - \epsilon = g_b$, i.e., both tasks are executed at the same speed. For any value of ϵ such that $0 < \epsilon \leq f_a - \frac{w_a + w_b}{t}$, $h(\epsilon) > 0$ and there is a strict gain in energy by decreasing the speed of T_a to $f_a - \epsilon$, and increasing the speed of T_b accordingly.

To conclude, we state that the new speeds of tasks $w_1, \ldots, w_k$ (that have been increased) remain always lower than the new speed of T_1^*, $\frac{w_1^*}{D}$ (that has been decreased), and therefore there is a strict gain in energy because the total execution time of T_1^* and the tasks of weights $w_1, \ldots, w_k$ remains constant. We can iteratively gain some time on p_2 by increasing the speed of a task with $f_i < f_{\texttt{rel}}$ up to $f_{\texttt{rel}}$ ($1 \leq i \leq k$), until task T_2^* fits on the processor, and at each step there is a strict gain in energy, hence the contradiction.

Finally, we have shown that it is strictly better to execute task T_2^* on processor p_2, and therefore T_1^* is executed alone on processor p_1, at a speed $\frac{w_1^*}{D} \geq f_{\texttt{rel}}$. ■

Next, we prove a lemma that will allow us to tackle the case where the load is *large enough* ($S > pDf_{\texttt{rel}}$), and we obtain a minimum on the approximation ratio of the deadline β.

Lemma 7.6. *For the problem* TRI-CRIT-INDEP *where each task T_i is such that $w_i < \max(\frac{S}{p}, Df_{rel})$, scheduling each task only once at speed $\max(f_{rel}, \frac{S}{pD})$ with the* LONGEST-PROCESSING-TIME *heuristic leads to a makespan of at most βD, with $\beta = \max\left(2 - \frac{3}{2p+1},\ 2 - \frac{p+2}{4p+2}\right)$.*

Note that we introduce $\max(\frac{S}{p}, Df_{\texttt{rel}})$ since the lemma is also used in the case $S \leq pDf_{\texttt{rel}}$. Also, since β is increasing with p and the bound is computed in fact for a number of processors smaller than the original one (some processors are dedicated to *big* tasks), the value of β computed with the total number of processors p is not smaller and it is possible to achieve a makespan of at most βD.

Proof. Let l_{lpt} be the maximal load of the processors after applying LONGEST-PROCESSING-TIME on the weights of the tasks. Let us find β such that $l_{\text{lpt}} \frac{pD}{S} \leq \beta D$: this means that within a time βD, we can schedule all tasks at speed $\frac{S}{pD}$, and therefore at speed $\max(f_{\texttt{rel}}, \frac{S}{pD})$, since the most loaded processor succeeds to be within the deadline βD.

Let l_{opt} be the maximal load of the processors in an optimal solution, and let T_i be the last task executed on the processor with the maximal load l_{lpt} by LONGEST-PROCESSING-TIME. We have either $w_i \leq l_{\text{opt}}/3$ or $w_i > l_{\text{opt}}/3$.

• If $w_i \leq l_{\text{opt}}/3$, we know that $l_{\text{opt}} \leq l_{\text{lpt}} \leq \left(\frac{4}{3} - \frac{1}{3p}\right) l_{\text{opt}}$, since LONGEST-PROCESSING-TIME is a $\left(\frac{4}{3} - \frac{1}{3p}\right)$-approximation [55]. We want to compare l_{opt} to S/p (average load). We consider the solution of LONGEST-PROCESSING-TIME. At the time when T_i was scheduled, all the processors were at least as loaded as the one on which T_i was scheduled, and hence we obtain a lower bound on S: $S \geq (p-1)(l_{\text{lpt}} - w_i) + l_{\text{lpt}}$. Furthermore, $l_{\text{lpt}} - w_i \geq \frac{2}{3} l_{\text{opt}}$ (because $l_{\text{lpt}} \geq l_{\text{opt}}$ and $w_i \leq l_{\text{opt}}/3$). Finally, $S \geq (p-1)\frac{2}{3} l_{\text{opt}} + l_{\text{opt}}$, which means that $l_{\text{opt}} \leq \frac{S}{p} \frac{3p}{2p+1}$, and $l_{\text{lpt}} \leq \left(\frac{4}{3} - \frac{1}{3p}\right) \frac{3p}{2p+1} \frac{S}{p} = \left(2 - \frac{3}{2p+1}\right) \frac{S}{p}$.

In this case, with $\beta = 2 - \frac{3}{2p+1}$, we can execute all the tasks at speed $\max(f_{\texttt{rel}}, \frac{S}{pD})$ within the deadline βD.

• If $w_i > l_{\text{opt}}/3$, it is known that LONGEST-PROCESSING-TIME is optimal for the execution time [55], i.e., $l_{\text{opt}} = l_{\text{lpt}}$, and we aim at finding an upper bound on l_{opt}. We assume in the following that tasks are numbered by non increasing weights.

If $w_i \geq \frac{S}{p}$, then we show that T_i is the only task executed on its processor (recall that T_i is the last task executed on the processor with the maximal load by LONGEST-PROCESSING-TIME). Indeed, there cannot be p tasks of weight at least $\frac{S}{p}$, hence $i < p$, and T_i is the first task scheduled on its processor. Moreover, if LONGEST-PROCESSING-TIME were to schedule another task on the processor of T_i, then this would mean that the $p-1$ other processors all have a load greater than w_i, and hence the total load would be greater than S. Then, since $w_i < \max(\frac{S}{p}, Df_{\texttt{rel}})$ and $w_i \geq \frac{S}{p}$, we have $w_i < Df_{\texttt{rel}}$ and we can execute each task at speed $f_{\texttt{rel}} = \max(f_{\texttt{rel}}, \frac{S}{pD})$ within a deadline D. Indeed, the maximal load is then w_i, by definition of T_i. Therefore, the result holds (with $\beta = 1$).

Now suppose that $w_i < \frac{S}{p}$. In that case, if T_i was the only task executed on its processor, then we would have $l_{\text{opt}} = l_{\text{lpt}} < \frac{S}{p}$, which is impossible since $S = \sum_{k=1}^{p} l_k \leq p l_{\text{opt}}$. Therefore, T_i is not the only task executed on its processor. A direct consequence of this fact is that $p+1 \leq i$. Indeed, LONGEST-PROCESSING-TIME schedules the p largest tasks on p distinct processors; since T_i is the last task scheduled on its processor, but not the only one, then T_i is not among the p first scheduled tasks. Also, there are only two tasks on the processor executing T_i, since $w_i > l_{\text{opt}}/3$ and the tasks scheduled before T_i have a weight at least equal to w_i. Finally, $p+1 \leq i \leq 2p$.

After scheduling task T_j on processor j for $1 \leq j \leq p$, LONGEST-PROCESSING-TIME schedules task T_{p+j} on processor $p-j+1$ for $1 \leq j \leq i-p$, and T_i is therefore scheduled on processor p_{2p-i+1}, together with task T_{2p-i+1}, and we have $w_i + w_{2p-i+1} = l_{\text{opt}}$. Note that because the w_j are sorted, $S \geq \sum_{j \leq i} w_j \geq i w_i$. We also have $w_{2p-i+1} < \frac{S}{p}$: indeed, when T_i was scheduled, the load of the p processors was at least equal to the load of the processor where T_{2p-i+1} was scheduled. Hence, w_{2p-i+1} cannot be greater than $\frac{S}{p}$. Then, since $w_{2p-i+1} = l_{\text{opt}} - w_i$, $w_i > l_{\text{opt}} - \frac{S}{p}$, and finally $l_{\text{opt}} - \frac{S}{p} < w_i \leq \frac{S}{i}$.

In order to find an upper bound on l_{opt}, we provide a lower bound to S, as a function of w_i:

$$
\begin{aligned}
S = \sum_{j=1}^{n} w_j \geq \sum_{j=1}^{i} w_j &= \sum_{j=1}^{2p-i+1} w_j + \sum_{j=2p-i+2}^{i} w_j \\
&\geq (2p-i+1) w_{2p-i+1} + (2(i-p)-1) w_i \\
&= (2p-i+1)(l_{\text{opt}} - w_i) + (2(i-p)-1) w_i \\
&= (2p-i+1) l_{\text{opt}} + (3i-4p-2) w_i = f(w_i).
\end{aligned}
$$

We then have $f'(w_i) = 3i - 4p - 2$, and we consider two cases.

If $f'(w_i) \geq 0$, then we have $i \geq \frac{4p+2}{3}$, and finally $S \geq i w_i > \frac{4p+2}{3} \left(l_{\text{opt}} - \frac{S}{p}\right)$. We can conclude that $l_{\text{opt}} < \frac{S}{p} \left(1 + \frac{3p}{4p+2}\right) = \frac{S}{p} \left(2 - \frac{p+2}{4p+2}\right)$.

Otherwise, $f'(w_i) < 0$ and f is a decreasing function of w_i, i.e., its minimum is reached when w_i is maximal, and $S \geq f(\frac{S}{i})$. Hence, $S \geq (2p - i + 1)l_{\text{opt}} + (3i - 4p - 2)\frac{S}{i}$. Since $i \leq 2p$, $2p - i + 1 > 0$ and

$$l_{\text{opt}} \leq \frac{S}{i}\left(\frac{i - 3i + 4p + 2}{2p - i + 1}\right) = \frac{2S}{i}.$$

Finally, since $i \geq p + 1$, $l_{\text{opt}} \leq \frac{2S}{p+1} = \frac{S}{p}\left(2 - \frac{2}{p+1}\right)$.

Overall, if $w_i > l_{\text{opt}}/3$, we have the bound

$$l_{\text{opt}} \leq \frac{S}{p} \times \max\left(2 - \frac{p+2}{4p+2}, 2 - \frac{2}{p+1}\right).$$

Therefore, for $\beta \geq \max\left(2 - \frac{p+2}{4p+2}, 2 - \frac{2}{p+1}\right)$, we can execute all the tasks on the processor of maximal load (and hence all the tasks) at speed $\max(f_{\texttt{rel}}, \frac{S}{pD})$ within the deadline βD in the case $w_i > l_{\text{opt}}/3$.

We can now conclude the proof of Lemma 7.6 by saying that for $\beta = \max\left(2 - \frac{3}{2p+1}, 2 - \frac{p+2}{4p+2}, 2 - \frac{2}{p+1}\right)$, i.e., $\beta = \max\left(2 - \frac{3}{2p+1}, 2 - \frac{p+2}{4p+2}\right)$, scheduling each task only once at speed $\max(f_{\texttt{rel}}, \frac{S}{pD})$ with the LONGEST-PROCESSING-TIME heuristic leads to a makespan of at most βD. ■

We are now ready to prove Theorem 7.3.

Proof of Theorem 7.3. First, thanks to Proposition 7.5, we know that the first step of the algorithm takes decisions that are identical to the optimal solution, and therefore these tasks that are executed once, alone on their processor, have the same energy consumption as the optimal solution and the same deadline. We can therefore safely ignore them in the remainder of the proof, and consider that for each task T_i, $w_i < \max(\frac{S}{p}, Df_{\texttt{rel}})$.

In the case where $S > pDf_{\texttt{rel}}$, we use the fact that $S(\frac{S}{pD})^2$ is a lower bound on the energy (Proposition 7.3). Each task is executed once at speed $\max(f_{\texttt{rel}}, \frac{S}{pD}) = \frac{S}{pD}$, and therefore the energy consumption is equal to the lower bound $S(\frac{S}{pD})^2$. The bound on the deadline is obtained by applying Lemma 7.6.

We now focus on the case $S \leq pDf_{\texttt{rel}}$. Therefore, in the following, $\max(\frac{S}{pD}, f_{\texttt{rel}}) = f_{\texttt{rel}}$. The algorithm runs the FPTAS on a linear chain of tasks with deadline pD, and ε as defined in Equation (7.3). The FPTAS returns a solution on the linear chain with an energy consumption E_{FPTAS} such that $E_{\text{FPTAS}} \leq (1 + \varepsilon)^2 E_{\text{chain}}$, where E_{chain} is the optimal energy consumption for TRI-CRIT-CHAIN with deadline pD on a single processor. According to Proposition 7.4, since the solution for the linear chain is a lower bound, the optimal solution of TRI-CRIT-INDEP is such that $E_{opt} \geq E_{\text{chain}}$.

For each task T_i, let f_i^{chain} be the speed of its execution returned by the FPTAS for TRI-CRIT-CHAIN. Note that in case of re-execution, then both executions occur at the same speed (Lemma 7.3). We now consider the TRI-CRIT-INDEP problem with the set of tasks $\tilde{V}$: for each task T_i, $\tilde{T}_i \in \tilde{V}$ and its weight is $\tilde{w}_i = w_i \frac{f_{\texttt{rel}}}{f_i^{\text{chain}}}$; moreover, if T_i is re-executed, we add two copies of $\tilde{T}_i$ in $\tilde{V}$. Then, $\sum_{\tilde{T}_i \in \tilde{V}} \frac{\tilde{w}_i}{f_{\texttt{rel}}} = pD$ by definition of the solution of TRI-CRIT-CHAIN.

Let $\beta = \max(2 - \frac{3}{2p+1}, 2 - \frac{p+2}{4p+2})$ be the relaxation on the deadline that we have from Lemma 7.6. The goal is to map all the tasks of $\tilde{V}$ at speed $f_{\texttt{rel}}$ within the deadline βD, which amounts to mapping the original tasks at the speeds assigned by the FPTAS:

- If there are tasks $\tilde{T}_i$ such that $\frac{\tilde{w}_i}{f_{\texttt{rel}}} > \beta D$, we execute them at speed $\frac{\tilde{w}_i}{\beta D}$ alone on an unused processor, so that they reach exactly the deadline βD. Note that in this case, the energy consumption of the algorithm becomes greater than E_{FPTAS}, since we execute these tasks faster than the FPTAS to fit on the processor.
- Tasks $\tilde{T}_i$ such that $D \leq \frac{\tilde{w}_i}{f_{\texttt{rel}}} \leq \beta D$ are executed alone on an unused processor at speed $f_{\texttt{rel}}$.
- For the remaining tasks and processors, we use LONGEST-PROCESSING-TIME as in Lemma 7.6. Since the previous tasks take a time of at least D in the solution of the FPTAS, and they are mapped alone on a processor, we can safely remove them and apply the lemma. Note that the number of processors may now be smaller than p, hence leading to a smaller bound β.

In the end, all tasks are mapped within the deadline βD (where β is computed with the original number of processors). There remains to check the energy consumption of the solution returned by this algorithm.

If all tasks are such that $\tilde{w}_i \leq \beta D f_{\texttt{rel}}$,
$E_{algo} = E_{\text{FPTAS}} \leq (1+\varepsilon)^2 E_{\text{chain}} \leq (1+\varepsilon)^2 E_{opt}$.
According to Equation (7.3), $\varepsilon \leq \frac{1}{3\beta^2}$, and therefore

$$E_{algo} \leq \left(1 + \frac{2}{3\beta^2} + \frac{1}{9\beta^4}\right) E_{opt} \leq \left(1 + \frac{1}{\beta^2}\right) E_{opt}.$$

Otherwise, let $\tilde{V}'$ be the set of tasks $\tilde{T}_i$ such that $\tilde{w}_i > \beta D f_{\texttt{rel}}$. For $\tilde{T}_i \in \tilde{V}'$, $w_i > \beta D f_i^{\text{chain}}$. Since $w_i < D f_{\texttt{rel}}$ (larger tasks have been processed in the first step of the algorithm), we have $f_i^{\text{chain}} < f_{\texttt{rel}}$. This means that T_i belongs to the set of the tasks that are re-executed by the FPTAS. Hence, since we enforced an additional constraint, we have $w_i < \frac{1}{\sqrt{2}} D f_{\texttt{rel}}$. The least energy consumed for this task by any solution to TRI-CRIT-INDEP is therefore obtained when re-executing task T_i on two distinct processors at speed $\frac{w_i}{D}$, in order to fit within the deadline D. Task T_i appears two times in $\tilde{V}'$, and we let $\tilde{E}$ be the minimum energy consumption required in the optimal solution for tasks of $\tilde{V}'$: $\tilde{E} = \sum_{\tilde{T}_i \in \tilde{V}'} w_i \left(\frac{w_i}{D}\right)^2$.

The algorithm leads to the same energy consumption as the FPTAS except for the tasks of $\tilde{V}'$ that are removed from the set X of replicated tasks, and that are executed at speed $\frac{w_i}{\beta D}$:

$$E_{algo} = (S-X) f_{\texttt{rel}}^2 + (2X - \textstyle\sum_{\tilde{T}_i \in \tilde{V}'} w_i) f_{\texttt{re-ex}}^2 + \sum_{\tilde{T}_i \in \tilde{V}'} w_i \left(\frac{w_i}{\beta D}\right)^2.$$

Since $E_{\text{FPTAS}} = (S-X) f_{\texttt{rel}}^2 + 2X f_{\texttt{re-ex}}^2$, we obtain

$$E_{algo} = E_{\text{FPTAS}} + \frac{1}{\beta^2}\tilde{E} - \textstyle\sum_{\tilde{T}_i \in \tilde{V}'} w_i f_{\texttt{re-ex}}^2.$$

Furthermore, $\tilde{E} \leq E_{opt}$ since it considers only the optimal energy consumption of a subset of tasks. We have $E_{\text{FPTAS}} \leq (1+\varepsilon)^2 E_{opt}$, and from Proposition 7.1, it is easy to see that $E_{\text{FPTAS}} \leq S f_{\texttt{rel}}^2$, i.e., E_{FPTAS} is smaller than the energy of every task executed once at speed $f_{\texttt{rel}}$. Hence, $E_{\text{FPTAS}} \leq (1+\varepsilon)^2 \min(E_{opt}, S f_{\texttt{rel}}^2)$, and since $\varepsilon < 1$, $(1+\varepsilon)^2 \leq 1 + 3\varepsilon$. Finally, $E_{\text{FPTAS}} \leq E_{opt} + 3\varepsilon S f_{\texttt{rel}}^2$. Thanks to Equation (7.3), $3\varepsilon S f_{\texttt{rel}}^2 \leq 2 w_{min} f_{\text{min}}^2 \leq \sum_{\tilde{T}_i \in \tilde{V}'} w_i f_{\texttt{re-ex}}^2$ (note that there are at least two tasks in $\tilde{V}'$, because tasks are duplicated), and finally

$$\begin{aligned} E_{algo} &\leq E_{opt} + 3\varepsilon S f_{\texttt{rel}}^2 + \frac{1}{\beta^2} E_{opt} - \textstyle\sum_{\tilde{T}_i \in \tilde{V}'} w_i f_{\texttt{re-ex}}^2 \\ &\leq \left(1 + \frac{1}{\beta^2}\right) E_{opt}. \end{aligned}$$

To conclude, we point out that this algorithm is polynomial in the size of the input and in $\frac{1}{\varepsilon}$. ■

We can improve the approximation ratio on the energy for large values of p. The idea is to avoid the case in which tasks are replicated by the chain but are not fitting within βD because the speed at which they are re-executed is too small. To do so, we fix a value $\varepsilon^* = \Theta\left(\frac{1}{p}\right)$, such that $0 < \varepsilon^* < 1$ for $p \geq 24$. The variant of the algorithm is used only when $p \geq 24$ (after scheduling the big tasks). The algorithm decides that the load is large enough when $S > pDf_{\texttt{rel}}\frac{1}{1+\varepsilon^*}$, leading to a $((1+\varepsilon^*)^2, \beta)$-approximation in this case. In the other case ($S \leq pDf_{\texttt{rel}}\frac{1}{1+\varepsilon^*}$), it is possible to prove that when there are tasks such that $\frac{\tilde{w}_i}{f_{\texttt{rel}}} > \beta D$, then necessarily all tasks are re-executed. Next we apply Theorem 7.1 while fixing values for the $f_{\text{inf},i}$'s, so as to obtain in polynomial time the optimal solution with new execution speeds, that can all be scheduled within βD using Lemma 7.6. Details can be found in the research report [RR1].

7.5 Conclusion

In this chapter, we have designed efficient approximation algorithms for the tri-criteria energy, reliability, and makespan problem, using replication and re-execution to increase the reliability, and dynamic voltage and frequency scaling to decrease the energy consumption. Because of the antagonistic relationship between energy and reliability, this tri-criteria problem is much more challenging than the standard bi-criteria problem, which aims at minimizing the energy consumption with a bound on the makespan, without accounting for a constraint on the reliability of tasks.

We have tackled two classes of applications. For linear chains of tasks, we propose a fully polynomial-time approximation scheme. However, we show that there exists no constant factor approximation algorithm for independent tasks, unless P=NP, and we are able in this case to propose an approximation algorithm with a relaxation on the makespan constraint.

As future work, it may be possible to improve the deadline relaxation by using an FPTAS to schedule independent tasks [5] rather than LONGEST-PROCESSING-TIME [55]. Also, an open problem is to find approximation algorithms for the tri-criteria problem with an arbitrary graph of tasks. Even though efficient heuristics have been designed with re-execution of tasks (but no replication) in Chapter 6, it is not clear how to derive approximation ratios from these heuristics. It would be interesting to design efficient algorithms using replication and re-execution for the general case, and to prove approximation ratios on these algorithms. A first step would be to tackle fork and fork-join graphs, inspired by the study on independent tasks. Finally, more sophisticated models for reliability could also be considered, for instance to guarantee a global reliability constraint or to authorize more than one backup task.

Chapter 8

Energy-aware checkpointing of divisible tasks with soft or hard deadlines

8.1 Introduction

Divisible load scheduling has been extensively studied in the past years [13, 41]. For divisible applications, the computational workload can be divided into an arbitrary number of chunks, whose sizes can be freely chosen by the user. Such applications occur for instance in the processing of very large data files, e.g., signal processing, linear algebra computation, or DNA sequencing. Traditionally, the goal is to minimize the makespan of the application, i.e., the total execution time.

Given a workload W, we need to decide how many chunks to use, and of which sizes. Using more chunks leads to a higher checkpoint cost, but smaller chunks imply less computation loss (and less re-execution) when a failure occurs. We assume that a chunk can fail only once, i.e., we re-execute each chunk at most once. Indeed, the probability that a fault would strike during both the first execution and the re-execution is negligible. We discuss the accuracy of this assumption in Section 8.3.

Due to the probabilistic nature of failure hits, it is natural to study the expectation $\mathbb{E}(E)$ of the energy consumption, because it represents the average cost over many executions. As for the bound D on execution time (the deadline), there are two relevant scenarios: either we enforce that this bound is a *soft deadline* to be met in expectation, or we enforce that this bound is a *hard deadline* to be met in the worst case. The former scenario corresponds to flexible environment where task deadlines can be viewed as average response times [24], while the latter scenario corresponds to real-time environments where task deadlines are always strictly enforced [115]. In both scenarios, we have to determine the number of chunks, their sizes, and the speed at which to execute (and possibly re-execute) every chunk.

Our first contribution is to formalize this important multi-objective problem. The general problem consists of finding n, the number of chunks, as well as the speeds for the execution and the re-execution of each chunk, both for soft and hard deadlines. We identify and discuss two important sub-cases that help tackling the most general problem instance: (i) a single chunk (the task is atomic); and (ii) re-execution speed is always identical to first execution speed. The second contribution is a comprehensive study of all problem instances; for each instance, we propose either an exact solution, or a function that can be optimized numerically. We also analytically prove the accuracy of our model that enforces a single re-execution per chunk. We then compare the different models through an extensive set of experiments. We compare the optimal energy consumption under various models with a set of different parameters. It turns out that when λ is small, it is sufficient to restrict the study to a single chunk, while when λ increases, it is better to use multiple chunks and different re-execution speeds.

The rest of the chapter is organized as follows. The model and the optimization problems are formal-

ized in Section 8.2. We discuss the accuracy of the model in Section 8.3. We first focus in Section 8.4 on the simpler case of an atomic task, i.e., with a single chunk. The general problem with multiple chunks, where we need to decide for the number of chunks and their sizes, is discussed in Section 8.5. In Section 8.6, we report several experiments to assess the differences between the models, and the relative gain due to chunking or to using different speeds for execution and re-execution. Finally, we provide some concluding remarks and future research directions in Section 8.7.

8.2 Framework

Given a workload W, the problem is to divide W into a number of chunks and to decide at which speed each chunk is executed. In case of a transient failure during the execution of one chunk, this chunk is re-executed, possibly at a different speed. We formalize the model in Section 8.2.1, and then different variants of the optimization problem are defined in Section 8.2.2. Table 8.1 summarizes the main notations.

W	total amount of work
s	processor speed for first execution
σ	processor speed for re-execution
T_C	checkpointing time
E_C	energy spent for checkpointing

Table 8.1: List of main notations.

8.2.1 Model

Consider first the case of a single chunk (or atomic task) of size W, denoted as SINGLECHUNK. We execute this chunk on a processor that can run at several speeds. We assume continuous speeds, i.e., the speed of execution can take an arbitrary positive real value. The execution is subject to failure, and resilience is provided through the use of checkpointing. The overhead induced by checkpointing is twofold: execution time T_C, and energy consumption E_C.

We assume that failures strike with uniform distribution, hence the probability that a failure occurs during an execution is linearly proportional to the length of this execution. Consider the first execution of a task of size W executed at speed s: the execution time is $T_{\text{exec}} = W/s + T_C$, hence the failure probability is $P_{\text{fail}} = \lambda T_{\text{exec}} = \lambda(W/s+T_C)$, where λ is the instantaneous failure rate. If there is indeed a failure, we re-execute the task at speed σ (which may or may not differ from s); the re-execution time is then $T_{\text{reexec}} = W/\sigma + T_C$ so that the expected execution time is

$$\begin{aligned}\mathbb{E}(T) &= T_{\text{exec}} + P_{\text{fail}} T_{\text{reexec}} \\ &= (W/s + T_C) + \lambda(W/s + T_C)(W/\sigma + T_C)\ . \end{aligned} \tag{8.1}$$

Similarly, the worst-case execution time is

$$\begin{aligned}T_{wc} &= T_{\text{exec}} + T_{\text{reexec}} \\ &= (W/s + T_C) + (W/\sigma + T_C)\ . \end{aligned} \tag{8.2}$$

Remember that we assume success after re-execution, so we do not account for second and more re-executions. Along the same line, we could spare the checkpoint after re-executing the last task in a

series of tasks, but this unduly complicates the analysis. In Section 8.3, we show that this model with only a single re-execution is accurate up to second order terms when compared to the model with an arbitrary number of failures that follows an Exponential distribution of parameter λ.

What is the expected energy consumed during execution? The energy consumed during the first execution at speed s is $Ws^2 + E_C$, where E_C is the energy consumed during a checkpoint. The energy consumed during the second execution at speed σ is $W\sigma^2 + E_C$, and this execution takes place with probability $P_{\text{fail}} = \lambda T_{\text{exec}} = \lambda(W/s + T_C)$, as before. Hence the expectation of the energy consumed is

$$\mathbb{E}(E) = (Ws^2 + E_C) + \lambda\,(W/s + T_C)\left(W\sigma^2 + E_C\right). \tag{8.3}$$

With multiple chunks (MULTIPLECHUNKS model), the execution times (worst case or expected) are the sum of the execution times for each chunk, and the expected energy is the sum of the expected energy for each chunk (by linearity of expectations).

We point out that the failure model is coherent with respect to chunking. Indeed, assume that a divisible task of weight W is split into two chunks of weights w_1 and w_2 (where $w_1 + w_2 = W$). Then the probability of failure for the first chunk is $P^1_{\text{fail}} = \lambda(w_1/s + T_C)$ and that for the second chunk is $P^2_{\text{fail}} = \lambda(w_2/s + T_C)$. The probability of failure $P_{\text{fail}} = \lambda(W/s + T_C)$ with a single chunk differs from the probability of failure with two chunks only because of the extra checkpoint that is taken; if $T_C = 0$, they coincide exactly. If $T_C > 0$, there is an additional risk to use two chunks, because the execution lasts longer by a duration T_C. Of course this is the price to pay for a shorter re-execution time in case of failure: Equation (8.1) shows that the expected re-execution time is $P_{\text{fail}} T_{\text{reexec}}$, which is quadratic in W. There is a trade-off between having many small chunks (many T_C to pay, but small re-execution cost) and a few larger chunks (fewer T_C, but increased re-execution cost).

8.2.2 Optimization problems

The optimization problem is stated as follows: given a deadline D and a divisible task whose total computational load is W, the problem is to partition the task into n chunks of size w_i, where $\sum_{i=1}^{n} w_i = W$, and choose for each chunk an execution speed s_i and a re-execution speed σ_i in order to minimize the expected energy consumption:

$$\mathbb{E}(E) = \sum_{i=1}^{n} (w_i s_i^2 + E_C) + \lambda \left(\frac{w_i}{s_i} + T_C\right)\left(w_i \sigma_i^2 + E_C\right),$$

subject to the constraint that the deadline is met either in expectation or in the worst case:

$$\begin{array}{ll} \text{EXPECTED-DEADLINE} & \mathbb{E}(T) = \sum_{i=1}^{n} \left(\frac{w_i}{s_i} + T_C + \lambda \left(\frac{w_i}{s_i} + T_C\right)\left(\frac{w_i}{\sigma_i} + T_C\right)\right) \leq D \\ \text{HARD-DEADLINE} & T_{wc} = \sum_{i=1}^{n} \left(\frac{w_i}{s_i} + T_C + \frac{w_i}{\sigma_i} + T_C\right) \leq D \end{array}$$

The unknowns are the number of chunks n, the sizes of these chunks w_i, the speeds for the first execution s_i and the speeds for the second execution σ_i. We consider two variants of the problem, depending upon re-execution speeds:

- SINGLESPEED : in this simpler variant, the re-execution speed is always the same as the speed chosen for the first execution. We then have to determine a single speed for each chunk: $\sigma_i = s_i$ for all i.

- MULTIPLESPEEDS : in this more general variant, the re-execution speed is freely chosen, and there are two different speeds to determine for each chunk.

We also consider the variant with a single chunk (SINGLECHUNK), i.e., the task is atomic and we only need to decide for its execution speed (in the SINGLESPEED model), or for its execution and re-execution speeds (in the MULTIPLESPEEDS model). We start the study in Section 8.4 with this simpler problem.

8.3 Accuracy of the model

In this section, we discuss the accuracy of this model, which accounts for a single re-execution. We compare the expressions of the expected deadline and energy (in Equations (8.1) and (8.3)) to those obtained when adopting the more advanced model where an arbitrary number of Exponentially distributed failures can strike during execution and re-execution. We only deal with soft deadlines here, because no hard deadline can be enforced for the model with Exponentially distributed failures (the execution time of a chunk can be arbitrarily large, although such an event has low probability to occur).

Assume that failures are distributed using an Exponential distribution of parameter λ: the probability of failure during a time interval of length t is $P_{\text{fail}} = 1 - e^{-\lambda t}$. Consider a single task of size W that we first execute at speed s. If we detect a transient failure at the end of the execution, we re-execute the task until success, using speed σ at each of these new attempts. To the best of our knowledge, the expressions for $\mathbb{E}(T)$ and $\mathbb{E}(E)$ are unknown for this model, and we establish them below:

Proposition 8.1. *With an arbitrary number of Exponentially distributed failures and one single task of size W,*

$$\mathbb{E}(T) = W/s + T_C \quad + e^{\lambda(W/\sigma+T_C)} \left(1 - e^{-\lambda(W/s+T_C)}\right) (W/\sigma + T_C) \tag{8.4}$$

$$\mathbb{E}(E) = Ws^2 + E_C \quad + e^{\lambda(W/\sigma+T_C)} \left(1 - e^{-\lambda(W/s+T_C)}\right) (W\sigma^2 + E_C) \tag{8.5}$$

Proof. With an Exponential distribution, Equation (8.1) can be rewritten as $\mathbb{E}(T) = T_{\text{exec}} + P_{\text{fail}}\mathbb{E}(T_{\text{reexec}})$, where $T_{\text{exec}} = W/s + T_C$ and $P_{\text{fail}} = 1 - e^{-\lambda(W/s+T_C)}$. Since all re-executions are done at speed σ, the expectation of the re-execution time obeys the following equation:

$$\mathbb{E}(T_{\text{reexec}}) = (W/\sigma + T_C) + \left(1 - e^{-\lambda(W/\sigma+T_C)}\right) \mathbb{E}(T_{\text{reexec}})$$

We use the memoryless property of the Exponential distribution here: after a failure, the expectation of the time to re-execute the task is exactly the same as before the failure. This leads to $\mathbb{E}(T_{\text{reexec}}) = e^{\lambda(W/\sigma+T_C)}(W/\sigma + T_C)$. Reporting in the first equation, we end up with Equation (8.4). The expression of the expected energy consumption (Equation (8.5)) is derived using the same line of reasoning. ■

Proposition 8.2. *With an arbitrary number of Exponentially distributed failures and one single task of size W, when $\lambda \to 0$,*

$$\mathbb{E}(T) = (W/s + T_C) + \lambda(W/s + T_C)(W/\sigma + T_C) + O\left(\lambda^2\right) \tag{8.6}$$

$$\mathbb{E}(E) = (Ws^2 + E_C) + \lambda(W/s + T_C)(W\sigma^2 + E_C) + O\left(\lambda^2\right) \tag{8.7}$$

Proof. The first-order Taylor expansion of $x \mapsto e^x$ around 0 gives:

$$\mathbb{E}(T) \underset{\lambda\to 0}{=} (W/s + T_C) + \left(1 + \lambda(W/\sigma + T_C) + O\left(\lambda^2(W/\sigma + T_C)^2\right)\right) \times \left(\lambda(W/s + T_C) + O\left(\lambda^2(W/s + T_C)^2\right)\right)(W/\sigma + T_C)$$

Hence,

$$\mathbb{E}(T) \underset{\lambda\to 0}{=} (W/s + T_C) + \left(\lambda(W/s + T_C) + O\left(\lambda^2\right)\right)(W/\sigma + T_C)$$
$$\mathbb{E}(T) \underset{\lambda\to 0}{=} (W/s + T_C) + \lambda(W/s + T_C)(W/\sigma + T_C) + O\left(\lambda^2\right).$$

Again, the energy formula is built using the same rationale. ■

As a consequence of Proposition 8.2, the formulas that we consider with one single re-execution (Equations (8.1) and (8.3)) are accurate up to second order terms when compared to the model with an arbitrary number of Exponential failures. Note that this result is not obvious, because we drop a potentially arbitrarily large number of re-executions in the linear model with at most one re-execution. Furthermore, the result extends naturally when considering a divisible task and MULTIPLECHUNKS, since the result holds for each chunk, and by summation, one single re-execution of each chunk is accurate up to second order terms.

8.4 With a single chunk

In this section, we consider the case of a single chunk, or equivalently of an atomic task: given a non-divisible workload W and a deadline D, find the values of s and σ that minimize

$$\mathbb{E}(E) = (Ws^2 + E_C) + \lambda\left(\frac{W}{s} + T_C\right)(W\sigma^2 + E_C)$$

subject to

$$\mathbb{E}(T) = \left(\frac{W}{s} + T_C\right) + \lambda\left(\frac{W}{s} + T_C\right)\left(\frac{W}{\sigma} + T_C\right) \leq D$$

in the EXPECTED-DEADLINE model, and subject to

$$\frac{W}{s} + T_C + \frac{W}{\sigma} + T_C \leq D$$

in the HARD-DEADLINE model. We first deal with the SINGLESPEED model, where we enforce $\sigma = s$, before moving on to the MULTIPLESPEEDS model.

8.4.1 Single speed model

In this section, we express $\mathbb{E}(E)$ as functions of the speed s. That is, $\mathbb{E}(E)(s) = (Ws^2 + E_C)(1 + \lambda(W/s + T_C))$. The following result is valid for both EXPECTED-DEADLINE and HARD-DEADLINE models.

Lemma 8.1. *$\mathbb{E}(E)$ is convex on $\mathbb{R}_+^\star$. It admits a unique minimum*

$$s^\star = \frac{\lambda W}{6(1+\lambda T_C)}\left(\frac{-(3\sqrt{3}\sqrt{27a^2 - 4a} - 27a + 2)^{1/3}}{2^{1/3}} - \frac{2^{1/3}}{(3\sqrt{3}\sqrt{27a^2 - 4a} - 27a + 2)^{1/3}} - 1\right) \tag{8.8}$$

where $a = \lambda E_C\left(\frac{2(1+\lambda T_C)}{\lambda W}\right)^2$.

Proof. Let us prove that $g(s) = \mathbb{E}(E)(s)$ is convex and admits a unique minimum: we have $g'(s) = s(2W(1+\lambda T_C)) + \lambda W^2 - \frac{\lambda W E_C}{s^2}$, $g''(s) = (2W(1+\lambda T_C)) + \frac{2\lambda W E_C}{s^3} > 0$. This function is strictly convex in $\mathbb{R}_+^\star$, and $g' \underset{0^+}{\to} -\infty$, $g' \underset{\infty}{\to} \infty$ thus there exist a unique minimum.

Let us find the minimum. For $s > 0$, we have:

$$g'(s) = 0 \Leftrightarrow \left(\frac{2(1+\lambda T_C)}{\lambda W}\right)^3 s^3 + \left(\frac{2(1+\lambda T_C)}{\lambda W}\right)^2 s^2 - \lambda E_C \left(\frac{2(1+\lambda T_C)}{\lambda W}\right)^2 = 0$$
$$\Leftrightarrow X^3 + X^2 - \lambda E_C \left(\frac{2(1+\lambda T_C)}{\lambda W}\right)^2 = 0 \quad \text{where } X = \frac{2(1+\lambda T_C)}{\lambda W} s$$

Using a computer algebra software, it is easy to show that the minimum is obtained at the value $s = s^\star$ given by Equation 8.8. ■

Expected deadline

In the SINGLESPEED EXPECTED-DEADLINE model, we denote $\mathbb{E}(T)(s) = (W/s + T_C)(1 + \lambda(W/s + T_C))$ the constraint on the execution time.

Lemma 8.2. *For any D, if $T_C + \lambda T_C^2 \geq D$, then there is no solution. Otherwise, the constraint on the execution time can be rewritten as $s \in \left[W\frac{1+2\lambda T_C + \sqrt{4\lambda D + 1}}{2(D - T_C(1+\lambda T_C))}, +\infty\right($.*

Proof. The function $s \mapsto \mathbb{E}(T)(s)$ is strictly decreasing and converges to $T_C + \lambda T_C^2$. Hence, if $T_C + \lambda T_C^2 \geq D$, then there is no solution. Else there exist a minimum speed s_0 such that, $\mathbb{E}(T)(s_0) = D$, and for all $s \geq s_0$, $\mathbb{E}(T)(s) \leq D$.

More precisely, $s_0 = W\frac{1+2\lambda T_C + \sqrt{4\lambda D + 1}}{2(D - T_C(1+\lambda T_C))}$: since there is a unique solution to $\mathbb{E}(T)(s) = D$, we can solve this equation in order to find s_0. ■

To simplify the following results, we define

$$s_0 = W\frac{1+2\lambda T_C + \sqrt{4\lambda D + 1}}{2(D - T_C(1+\lambda T_C))}. \tag{8.9}$$

Proposition 8.3. *In the* SINGLESPEED *model, it is possible to numerically compute the optimal solution for* SINGLECHUNK *as follows:*

1. *If $T_C + \lambda T_C^2 \geq D$, then there is no solution;*
2. *Else, the optimal speed is* $\max(s_0, s^\star)$.

Proof. This is a corollary of Lemma 8.1: because $s \mapsto \mathbb{E}(T)(s)$ is convex on $\mathbb{R}_+^\star$, then its restriction to the interval $[s_0, +\infty($ is also convex and admits a unique minimum:

- if $s^\star < s_0$, then $\mathbb{E}(T)\ (s)$ is increasing on $[s_0, +\infty($, then the optimal solution is s_0
- else, clearly the minimum is reached when $s = s^\star$.

The optimal solution is then $\max(s_0, s^\star)$. ■

Hard deadline

In the HARD-DEADLINE model, the bound on the execution time can be written as $2\left(\frac{W}{s}+T_C\right) \leq D$

Lemma 8.3. *In the* SINGLESPEED HARD-DEADLINE *model, for any D, if $2T_C \geq D$, then there is no solution. Otherwise, the constraint on the execution time can be rewritten as $s \in \left[\frac{W}{\frac{D}{2}-T_C}; +\infty\right($*

Proof. The constraint on the execution time is now $2\left(\frac{W}{s}+T_C\right) \leq D$. ■

Proposition 8.4. *In the* SINGLESPEED HARD-DEADLINE *model if $2T_C \geq D$, then there is no solution. Otherwise, let let $s^\star$ the solution indicated in Equation 8.8, then the minimum is reached when $s = \max\left(s^\star, \frac{W}{\frac{D}{2}-T_C}\right)$.*

Proof. The fact that there is no solution when $2T_C \geq D$ comes from Lemma 8.3. Otherwise, the result is obvious by convexity of the expected energy function. ■

8.4.2 Multiple speeds model

In this section, we consider the general MULTIPLESPEEDS model. We use the following notations:

$$\mathbb{E}(E)(s,\sigma) = (Ws^2 + E_C) + \lambda(W/s + T_C)(W\sigma^2 + E_C)$$

Let us first introduce a preliminary Lemma:

Lemma 8.4 (Convexity SINGLECHUNK)**.** *The problem of minimizing $A_0 + \alpha_0 x^2$ under the constraint $A_1 + \frac{\alpha_1}{x} \leq A_2$ where A_0, A_1, A_2 are constants and α_0, α_1 are positive constants is solved when x is minimum, that is when $A_1 + \frac{\alpha_1}{x} = A_2$.*

Proof. The function $A_0 + \alpha_0 x^2$ is strictly increasing, so it is is minimized when x is minimum. The function $A_1 + \frac{\alpha_1}{x}$ is strictly decreasing with $\lim_{x\to 0} = +\infty$, hence an upper bound is reached when x is minimum. With those two results, we can say that the constraint should be tight in order to solve our problem. ■

Expected deadline

The execution time in the MULTIPLESPEEDS EXPECTED-DEADLINE model can be written as

$$\mathbb{E}(T)(s,\sigma) = (W/s + T_C) + \lambda(W/s + T_C)(W/\sigma + T_C)$$

We start by giving a useful property, namely that the deadline is always tight in the MULTIPLESPEEDS EXPECTED-DEADLINE model:

Lemma 8.5. *In the* MULTIPLESPEEDS EXPECTED-DEADLINE *model, in order to minimize the energy consumption, the deadline should be tight.*

Proof. Considering s and W fixed, then $\mathbb{E}(T)(s,\sigma) = T_0 + \frac{\alpha}{\sigma} \leq D$, and $\mathbb{E}(E)(s,\sigma) = E_0 + \alpha\sigma^2$, where $T_0 = (W/s + E_C) + \lambda T_C(W/s + T_C)$, $E_0 = (Ws^2 + E_C) + \lambda E_C(W/s + T_C)$ and $\alpha = W(W/s + T_C)$ are constant. With Lemma 8.4 we conclude that the deadline should be tight. ■

This lemma allows us to express σ as a function of s:

$$\sigma = \frac{\lambda W}{\frac{D}{\frac{W}{s}+T_C} - (1+\lambda T_C)}.$$

Also we reduce the bi-criteria problem to the minimization problem of the single-variable function:

$$s \mapsto Ws^2 + E_C + \lambda\left(\frac{W}{s}+T_C\right)\left(W\left(\frac{\lambda W}{\frac{D}{\frac{W}{s}+T_C} - (1+\lambda T_C)}\right)^2 + E_C\right) \tag{8.10}$$

which can be solved numerically.

Hard deadline

In this model we have similar results as with EXPECTED-DEADLINE. The constraint on the execution time writes: $\frac{W}{s} + T_C + \frac{W}{\sigma} + T_C \leq D$. Another corollary of Lemma 8.4 is:

Lemma 8.6. *In the* MULTIPLESPEEDS EXPECTED-DEADLINE *model, in order to minimize the energy consumption, the deadline should be tight.*

This lemma allows us to express σ as a function of s:

$$\sigma = \frac{Ws}{(D-2T_C)s - W}$$

Finally, we reduce the bi-criteria problem to the minimization problem of the single-variable function:

$$s \mapsto Ws^2 + E_C + \lambda\left(\frac{W}{s}+T_C\right)\left(W\left(\frac{Ws}{(D-2T_C)s - W}\right)^2 + E_C\right) \tag{8.11}$$

which can be solved numerically.

8.5 Several chunks

In this section, we deal with the general problem of a divisible task of size W that can be split into an arbitrary number of chunks. We divide the task into n chunks of size w_i such that $\sum_{i=1}^{n} w_i = W$. Each chunk is executed once at speed s_i, and re-executed (if necessary) at speed σ_i. The problem is to find the values of n, w_i, s_i and σ_i that minimize

$$\mathbb{E}(E) = \sum_i \left(w_i s_i^2 + E_C\right) + \lambda \sum_i \left(\frac{w_i}{s_i} + T_C\right)\left(w_i \sigma_i^2 + E_C\right)$$

subject to

$$\sum_i \left(\frac{w_i}{s_i} + T_C\right) + \lambda \sum_i \left(\frac{w_i}{s_i} + T_C\right)\left(\frac{w_i}{\sigma_i} + T_C\right) \leq D$$

in the EXPECTED-DEADLINE model, and subject to

$$\sum_i \left(\frac{w_i}{s_i} + T_C\right) + \sum_i \left(\frac{w_i}{\sigma_i} + T_C\right) \leq D$$

in the HARD-DEADLINE model. We first deal with the SINGLESPEED model, where we enforce $\sigma_i = s_i$, before dealing with the MULTIPLESPEEDS model.

8.5.1 Single speed model

Expected deadline

In this section, we deal with the SINGLESPEED EXPECTED-DEADLINE model and consider that for all i, $\sigma_i = s_i$. Then:

$$\mathbb{E}(T)(\cup_i(w_i, s_i, s_i)) = \sum_i \left(\frac{w_i}{s_i} + T_C\right) + \lambda \sum_i \left(\frac{w_i}{s_i} + T_C\right)^2$$
$$\mathbb{E}(E)(\cup_i(w_i, s_i, s_i)) = \sum_i \left(w_i s_i^2 + E_C\right)\left(1 + \lambda\left(\frac{w_i}{s_i} + T_C\right)\right)$$

Theorem 8.1. *In the optimal solution to the problem with the* SINGLESPEED EXPECTED-DEADLINE *model, all n chunks are of equal size W/n and executed at the same speed s.*

Proof. Consider the optimal solution, and assume by contradiction that it includes two chunks w_1 and w_2, executed at speeds s_1 and s_2, where either $s_1 \neq s_2$, or $s_1 = s_2$ and $w_1 \neq w_2$. Let us assume without loss of generality that $\frac{w_1}{s_1} \geq \frac{w_2}{s_2}$.

We show that we can find a strictly better solution where both chunks have size $w = \frac{1}{2}(w_1 + w_2)$, and are executed at same speed s (to be defined later). The size and speed of the other chunks are kept the same. We will show that the execution time of the new solution is not larger than in the optimal solution, while its energy consumption is strictly smaller, hence leading to the contradiction.

We have seen that

$$\mathbb{E}(T)((w_1, s_1), (w_2, s_2)) = \frac{w_1}{s_1} + T_C + \frac{w_2}{s_2} + T_C + \lambda\left(\frac{w_1}{s_1} + T_C\right)^2 + \lambda\left(\frac{w_2}{s_2} + T_C\right)^2$$
$$\mathbb{E}(T)((w, s), (w, s)) = 2\left(\frac{w}{s} + T_C\right) + 2\lambda\left(\frac{w}{s} + T_C\right)^2.$$

Hence,

$$\mathbb{E}(T)((w_1, s_1), (w_2, s_2)) - \mathbb{E}(T)((w, s), (w, s)) = \left(\frac{w_1}{s_1} + \frac{w_2}{s_2} - \frac{2w}{s}\right) + \lambda\left(\left(\frac{w_1}{s_1}\right)^2 + \left(\frac{w_2}{s_2}\right)^2 - 2\left(\frac{w}{s}\right)^2\right).$$

Similarly, we know that:

$$\mathbb{E}(E)((w_1, s_1), (w_2, s_2)) = w_1 s_1^2 + E_C + w_2 s_2^2 + E_C + \lambda\left(\frac{w_1}{s_1} + T_C\right)\left(w_1 s_1^2 + E_C\right) + \lambda\left(\frac{w_2}{s_2} + T_C\right)\left(w_2 s_2^2 + E_C\right)$$
$$\mathbb{E}(E)((w, s), (w, s)) = 2\left(w s^2 + E_C\right) + 2\lambda\left(\frac{w}{s} + T_C\right)\left(w s^2 + E_C\right)$$

and deduce

$$\mathbb{E}(E)((w_1, s_1), (w_2, s_2)) - \mathbb{E}(E)((w, s), (w, s)) = \left(w_1 s_1^2 + w_2 s_2^2 - 2w s^2\right)(1 + \lambda T_C) + \lambda E_C\left(\frac{w_1}{s_1} + \frac{w_2}{s_2} - \frac{2w}{s}\right) + \lambda\left(w_1^2 s_1 + w_2^2 s_2 - 2w^2 s\right). \tag{8.12}$$

Let us now define

$$s_A = \frac{2w}{\frac{w_1}{s_1}+\frac{w_2}{s_2}} = \frac{w_1+w_2}{\frac{w_1}{s_1}+\frac{w_2}{s_2}}$$
$$s_B = \frac{\sqrt{2}w}{\left(\left(\frac{w_1}{s_1}\right)^2+\left(\frac{w_2}{s_2}\right)^2\right)^{1/2}} = \frac{w_1+w_2}{\left(2\left(\frac{w_1}{s_1}\right)^2+2\left(\frac{w_2}{s_2}\right)^2\right)^{1/2}}.$$

We then fix $s = \max(s_A, s_B)$. Then, since $s \geq s_A$, we have $\frac{w_1}{s_1}+\frac{w_2}{s_2}-\frac{2w}{s} \geq 0$, and since $s \geq s_B$, we have $\left(\frac{w_1}{s_1}\right)^2+\left(\frac{w_2}{s_2}\right)^2-2\left(\frac{w}{s}\right)^2 \geq 0$. This ensures that $\mathbb{E}(T)((w_1,s_1),(w_2,s_2))-\mathbb{E}(T)((w,s),(w,s)) \geq 0$.

Note that

$$\frac{(w_1+w_2)^2}{s_B^2}-\frac{(w_1+w_2)^2}{s_A^2} = 2\left(\frac{w_1}{s_1}\right)^2+2\left(\frac{w_2}{s_2}\right)^2-\left(\frac{w_1}{s_1}+\frac{w_2}{s_2}\right)^2 = \left(\frac{w_1}{s_1}-\frac{w_2}{s_2}\right)^2 \geq 0.$$

This means that $s_A \geq s_B$, hence $s = s_A$. To prove that $\mathbb{E}(E)((w_1,s_1),(w_2,s_2))-\mathbb{E}(E)((w,s),(w,s)) > 0$, we need to show that:

1. $w_1 s_1^2 + w_2 s_2^2 - 2ws^2 \geq 0$,
2. $\frac{w_1}{s_1}+\frac{w_2}{s_2}-\frac{2w}{s} \geq 0$,
3. $w_1^2 s_1 + w_2^2 s_2 - 2w^2 s \geq 0$,
4. and that one of the previous inequalities is strict.

Note that by definition of $s = s_A$, the second inequality holds.

Let us first show that $w_1 s_1^2 + w_2 s_2^2 - 2ws_A^2 \geq 0$.

$$\left(\frac{w_1}{s_1}+\frac{w_2}{s_2}\right)^2\left(w_1 s_1^2 + w_2 s_2^2 - (w_1+w_2)\left(\frac{w_1+w_2}{\frac{w_1}{s_1}+\frac{w_2}{s_2}}\right)^2\right)$$
$$= w_1^3 + w_1 w_2^2\left(\frac{s_1}{s_2}\right)^2 + 2\frac{w_1 w_2}{s_1 s_2} w_1 s_1^2 + w_2^3 + w_2 w_1^2\left(\frac{s_2}{s_1}\right)^2 + 2\frac{w_2 w_1}{s_1 s_2} w_2 s_2^2 - (w_1+w_2)^3$$
$$= w_1 w_2^2\left(\left(\frac{s_1}{s_2}\right)^2 + 2\frac{s_2}{s_1} - 3\right) + w_1^2 w_2\left(\left(\frac{s_2}{s_1}\right)^2 + 2\frac{s_1}{s_2} - 3\right)$$
$$= w_1 w_2^2 g\left(\frac{s_1}{s_2}\right) + w_1^2 w_2 g\left(\frac{s_2}{s_1}\right)$$

where $g : u \mapsto u^2 + \frac{2}{u} - 3$. It is easy to show that g is nonnegative on $\mathbb{R}_+^\star$: indeed, $g'(u) = \frac{2}{u^2}(u^3-1)$ is negative over $[0,1[$ and positive over $]1,\infty[$, and the unique minimum of g is $g(1) = 0$. We derive that $w_1 s_1^2 + w_2 s_2^2 - 2ws_A^2 \geq 0$.

Let us now show that $w_1^2 s_1 + w_2^2 s_2 - 2w^2 s \geq 0$**.** Remember that $2w = w_1 + w_2$.

$$\begin{aligned}
2\left(\frac{w_1}{s_1}+\frac{w_2}{s_2}\right)&\left(w_1^2 s_1 + w_2^2 s_2 - \frac{(w_1+w_2)^3}{2\left(\frac{w_1}{s_1}+\frac{w_2}{s_2}\right)}\right) \\
&= 2w_1^3 + 2w_1^2 w_2 \frac{s_1}{s_2} + 2w_2^3 + 2w_1 w_2^2 \frac{s_2}{s_1} - (w_1+w_2)^3 \\
&= w_1^3 + w_2^3 + w_1^2 w_2\left(2\frac{s_1}{s_2} - 3\right) + w_1 w_2^2\left(2\frac{s_2}{s_1} - 3\right)
\end{aligned}$$

Remember that we assumed without loss of generality that $\frac{w_1}{s_1} \geq \frac{w_2}{s_2}$.

$$\begin{aligned}
2\left(\frac{w_1}{s_1}+\frac{w_2}{s_2}\right)&\left(w_1^2 s_1 + w_2^2 s_2 - \frac{(w_1+w_2)^3}{2\left(\frac{w_1}{s_1}+\frac{w_2}{s_2}\right)}\right) \\
&\geq w_2^3\left(\left(\frac{s_1}{s_2}\right)^3 + 1 + \left(\frac{s_1}{s_2}\right)^2\left(2\frac{s_1}{s_2} - 3\right) + \frac{s_1}{s_2}\left(2\frac{s_2}{s_1} - 3\right)\right) \\
&\geq 3w_2^3\left(\left(\frac{s_1}{s_2}\right)^3 - \left(\frac{s_1}{s_2}\right)^2 - \frac{s_1}{s_2} + 1\right) \\
&\geq 3w_2^3\left(\left(\frac{s_1}{s_2} - 1\right)^2\left(\frac{s_1}{s_2} + 1\right)\right) \geq 0
\end{aligned}$$

Let us now conclude the proof: if $\frac{s_1}{s_2} \neq 1$, then the energy consumption of the optimal solution is strictly greater than the one from our solution which is a contradiction. Hence we must have $s_1 = s_2$, and $w_1 \neq w_2$ (in fact, since we assumed that $\frac{w_1}{s_1} \geq \frac{w_2}{s_2}$, we must have $w_1 > w_2$). Then we can refine the previous analysis, and obtain that $w_1^2 s_1 + w_2^2 s_2 - 2w^2 s > 0$: again, the optimal energy consumption is strictly greater than in our solution; this is the final contradiction and concludes the proof. ■

Thanks to this result, we know that the problem with n chunks can be rewritten as follows: find s such that

$$n\left(\frac{W}{ns} + T_C\right) + n\lambda\left(\frac{W}{ns} + T_C\right)^2 = \frac{W}{s} + nT_C + \frac{\lambda}{n}\left(\frac{W}{s} + nT_C\right)^2 \leq D$$

in order to minimize

$$n\left(\frac{W}{n}s^2 + E_C\right) + n\lambda\left(\frac{W}{ns} + T_C\right)\left(\frac{W}{n}s^2 + E_C\right) = \left(Ws^2 + nE_C\right)\left(1 + \frac{\lambda}{n}\left(\frac{W}{s} + nT_C\right)\right)$$

One can see that this reduces to the SINGLECHUNK problem with the SINGLESPEED model (Section 8.4.1) up to the following parameter changes:

- $\lambda \leftarrow \frac{\lambda}{n}$,
- $T_C \leftarrow nT_C$,
- $E_C \leftarrow nE_C$.

If the number of chunks n is given, we can express the minimum speed such that there is a solution with n chunks:

$$s_0(n) = W \frac{1 + 2\lambda T_C + \sqrt{4\frac{\lambda D}{n} + 1}}{2(D - nT_C(1 + \lambda T_C))}. \tag{8.13}$$

We can verify that when $D \leq nT_C(1 + \lambda n)$, there is no solution, hence obtaining an upper bound on n. Therefore, the two variables problem (with unknowns n and s) can be solved numerically.

Hard deadline

In the HARD-DEADLINE model, all results still hold, they are even easier to prove since we do not need to introduce a second speed.

Theorem 8.2. *In the optimal solution to the problem with the* SINGLESPEED HARD-DEADLINE *model, all n chunks are of equal size W/n and executed at the same speed s.*

Proof. The proof is similar to the one of Theorem 8.1, except we do not need to study the case where $s_B > s_A$. ■

8.5.2 Multiple speeds model

Expected deadline

In this section, we still deal with the problem of a divisible task of size W that we can split into an arbitrary number of chunks, but using the more general MULTIPLESPEEDS model. We start by proving that all re-execution speeds are equal:

Let us first introduce a preliminary Lemma:

Lemma 8.7 (Convexity MULTIPLECHUNKS)**.** *The problem of minimizing $A_0 + \alpha_0 x_0^2 + \alpha_1 x_1^2$ under the constraint $\frac{\alpha_0}{x_0} + \frac{\alpha_1}{x_1} \leq A_1$ where A_0 is a constant, and A_1, α_0, α_1 are positive constants, is solved when $x_0 = x_1$, and when the constraint is tight: $\frac{\alpha_0}{x_0} + \frac{\alpha_1}{x_1} = A_1$.*

Proof. First remark that when x_1 is fixed, then according to Lemma 8.4, the constraint should be tight. Hence this is true for the optimal solution (any optimal solution when the constraint is not tight can be improved by reducing one of the variables).

To prove the result now that we know that the constraint is tight, it suffices to replace in the function we wish to minimize, $x_0 = \frac{\alpha_0}{A_1 - \frac{\alpha_1}{x_1}}$. Differentiating $A_0 + \alpha_0 \times \left(\frac{\alpha_0}{A_1 - \frac{\alpha_1}{x_1}}\right)^2 + \alpha_1 x_1^2$ with respect to x_1 gives $-\frac{2\alpha_1\alpha_0^3}{x_1^2\left(A_1 - \frac{\alpha_1}{x_1}\right)^3} + 2\alpha_1 x_1$. Then we obtain that the equation is minimized (by differentiating again, we can see that the function is convex) when $-\frac{2\alpha_1\alpha_0^3}{x_1^2\left(A_1 - \frac{\alpha_1}{x_1}\right)^3} + 2\alpha_1 x_1 = 0$, that is $-x_0 + x_1 = 0$, hence the result. ■

Note that if A_1 is non positive, then there is no solution.

Lemma 8.8. *In the* MULTIPLESPEEDS *model, all re-execution speeds are equal in the optimal solution: $\exists \sigma, \forall i, \sigma_i = \sigma$, and the deadline is tight.*

Proof. This is a direct corollary of Lemma 8.7. If we consider the w_i and s_i to be fixed, then we can write $\mathbb{E}(T)(\cup_i(w_i, s_i, \sigma_i)) = T_0 + \sum_i \frac{\alpha_i}{\sigma_i}$, and $\mathbb{E}(E)(\cup_i(w_i, s_i, \sigma_i)) = E_0 + \sum_i \alpha_i\sigma_i^2$, where T_0, E_0 and α_i are constant. Assuming $D - T_0 > 0$ (otherwise there is no solution), we can apply Lemma 8.7, then the problem is minimized when the deadline is tight, and when for all i, $\sigma_i = \frac{\sum_i \alpha_i}{D-T_0}$. ■

We can now redefine

$$\mathbb{E}(T)(\cup_i(w_i, s_i, \sigma_i)) = T(\cup_i(w_i, s_i), \sigma)$$
$$\mathbb{E}(E)(\cup_i(w_i, s_i, \sigma_i)) = E(\cup_i(w_i, s_i), \sigma).$$

Theorem 8.3. *In the* MULTIPLESPEEDS *model, all chunks have the same size $w_i = \frac{W}{n}$, and are executed at the same speed s, in the optimal solution.*

Proof. We first prove that chunks are of equal size. Assume first, by contradiction, that the optimal solution has two chunks of different sizes, for instance $w_1 < w_2$. These chunks are executed at speeds s_1 and s_2. Thanks to Lemma 8.8, both chunks are re-executed at a same speed σ. We consider the solution with two chunks of size $w = \frac{1}{2}(w_1 + w_2)$, executed at a same speed s (to be defined later), and re-executed at speed σ (the value of the re-execution speed in the optimal solution). The size and speed of the other chunks are kept the same. We show that the execution time is not greater than in the optimal solution, while the energy consumption is strictly smaller, hence leading to the contradiction.

We have seen that

$$\begin{aligned}\mathbb{E}(T)((w_1, s_1), (w_2, s_2), \sigma) = \frac{w_1}{s_1} + T_C + \frac{w_2}{s_2} + T_C + \lambda\left(\frac{w_1}{s_1} + T_C\right)\left(\frac{w_1}{\sigma} + T_C\right)\\ + \lambda\left(\frac{w_2}{s_2} + T_C\right)\left(\frac{w_2}{\sigma} + T_C\right)\\ \mathbb{E}(T)((w, s), (w, s), \sigma) = 2\left(\frac{w}{s} + T_C\right) + 2\lambda\left(\frac{w}{s} + T_C\right)\left(\frac{w}{\sigma} + T_C\right)\end{aligned}$$

Hence,

$$\begin{aligned}\mathbb{E}(T)((w_1, s_1), (w_2, s_2), \sigma) - \mathbb{E}(T)((w, s), (w, s), \sigma) = (1 + \lambda T_C)\left(\frac{w_1}{s_1} + \frac{w_2}{s_2} - \frac{2w}{s}\right)\\ + \frac{\lambda}{\sigma}\left(\frac{w_1^2}{s_1} + \frac{w_2^2}{s_2} - \frac{2w^2}{s}\right)\end{aligned}$$

Similarly, we know that:

$$\begin{aligned}\mathbb{E}(E)((w_1, s_1), (w_2, s_2), \sigma) = w_1 s_1^2 + E_C + w_2 s_2^2 + E_C + \lambda\left(\frac{w_1}{s_1} + T_C\right)\left(w_1\sigma^2 + E_C\right)\\ + \lambda\left(\frac{w_2}{s_2} + T_C\right)\left(w_2\sigma^2 + E_C\right)\\ \mathbb{E}(E)((w, s), (w, s), \sigma) = 2\left(ws^2 + E_C\right) + 2\lambda\left(\frac{w}{s} + T_C\right)\left(w\sigma^2 + E_C\right),\end{aligned}$$

and deduce

$$\begin{aligned}\mathbb{E}(E)((w_1, s_1), (w_2, s_2), \sigma) - \mathbb{E}(E)((w, s), (w, s), \sigma) = \left(w_1 s_1^2 + w_2 s_2^2 - 2ws^2\right)\\ + \lambda E_C\left(\frac{w_1}{s_1} + \frac{w_2}{s_2} - \frac{2w}{s}\right) + \lambda\sigma^2\left(\frac{w_1^2}{s_1} + \frac{w_2^2}{s_2} - \frac{2w^2}{s}\right).\end{aligned} \tag{8.14}$$

Let us now define

$$s_A = \frac{2w}{\frac{w_1}{s_1} + \frac{w_2}{s_2}} = \frac{w_1 + w_2}{\frac{w_1}{s_1} + \frac{w_2}{s_2}}$$
$$s_B = \frac{2w^2}{\frac{w_1^2}{s_1} + \frac{w_2^2}{s_2}} = \frac{1}{2}\frac{(w_1 + w_2)^2}{\frac{w_1^2}{s_1} + \frac{w_2^2}{s_2}}.$$

We then fix $s = \max(s_A, s_B)$. Then, since $s \geq s_A$, we have $\frac{w_1}{s_1} + \frac{w_2}{s_2} - \frac{2w}{s} \geq 0$, and since $s \geq s_B$, we have $\frac{w_1^2}{s_1} + \frac{w_2^2}{s_2} - \frac{2w^2}{s} \geq 0$. This ensures that $\mathbb{E}(T)((w_1, s_1), (w_2, s_2), \sigma) - \mathbb{E}(T)((w, s), (w, s), \sigma) \geq 0$. To prove that $\mathbb{E}(E)((w_1, s_1), (w_2, s_2), \sigma) - \mathbb{E}(E)((w, s), (w, s), \sigma) \geq 0$, there remains to show that $w_1 s_1^2 + w_2 s_2^2 - 2ws^2 \geq 0$.

Let us first suppose that $s_A > s_B$. Then we have $s = s_A$, and let us show that $w_1 s_1^2 + w_2 s_2^2 - 2ws_A^2 \geq 0$:

$$\begin{aligned}
&\left(\frac{w_1}{s_1} + \frac{w_2}{s_2}\right)^2 \left(w_1 s_1^2 + w_2 s_2^2 - (w_1 + w_2)\left(\frac{w_1 + w_2}{\frac{w_1}{s_1} + \frac{w_2}{s_2}}\right)^2\right) \\
&\quad = w_1^3 + w_1 w_2^2 \left(\frac{s_1}{s_2}\right)^2 + 2\frac{w_1 w_2}{s_1 s_2} w_1 s_1^2 + w_2^3 + w_2 w_1^2 \left(\frac{s_2}{s_1}\right)^2 + 2\frac{w_2 w_1}{s_1 s_2} w_2 s_2^2 - (w_1 + w_2)^3 \\
&\quad = w_1 w_2^2 \left(\left(\frac{s_1}{s_2}\right)^2 + 2\frac{s_2}{s_1} - 3\right) + w_1^2 w_2 \left(\left(\frac{s_2}{s_1}\right)^2 + 2\frac{s_1}{s_2} - 3\right) \\
&\quad = w_1 w_2^2 g\left(\frac{s_1}{s_2}\right) + w_1^2 w_2 g\left(\frac{s_2}{s_1}\right)
\end{aligned}$$

where $g : u \mapsto u^2 + \frac{2}{u} - 3$. We know from the proof of Theorem 8.1 that g is positive on $\mathbb{R}_+^\star$, hence $w_1 s_1^2 + w_2 s_2^2 - 2ws_A^2 \geq 0$.

Finally, since $s > s_B$, we have $\frac{w_1^2}{s_1} + \frac{w_2^2}{s_2} - \frac{2w^2}{s} > 0$, and all other terms of $\mathbb{E}(E)((w_1, s_1), (w_2, s_2), \sigma) - \mathbb{E}(E)((w, s_A), (w, s_A), \sigma)$ are non-negative, hence proving that the new solution is strictly better than the optimal one, and leading to a contradiction.

Let us now suppose that $s_A \leq s_B$. Then we have $s = s_B$. Moreover, we have $(w_2 - w_1)(\frac{w_2}{s_2} - \frac{w_1}{s_1}) \leq 0$ (this comes directly from $s_A \leq s_B$), and since we assume that $w_2 > w_1$, $\frac{w_2}{s_2} - \frac{w_1}{s_1} \leq 0$. Let us show that $w_1 s_1^2 + w_2 s_2^2 - 2ws_B^2 > 0$:

$$\begin{aligned}
&4\left(\frac{w_1^2}{s_1} + \frac{w_2^2}{s_2}\right)^2 \left(w_1 s_1^2 + w_2 s_2^2 - (w_1 + w_2)\left(\frac{1}{2}\frac{(w_1 + w_2)^2}{\frac{w_1^2}{s_1} + \frac{w_2^2}{s_2}}\right)^2\right) \\
&\quad = 4w_1^5 + 8w_1^3 w_2^2 \frac{s_1}{s_2} + 4w_1 w_2^4 \left(\frac{s_1}{s_2}\right)^2 + 4w_2^5 + 8w_1^2 w_2^3 \frac{s_2}{s_1} + 4w_1^4 w_2 \left(\frac{s_2}{s_1}\right)^2 - (w_1 + w_2)^5 \\
&\quad = 3\left(w_1^5 + w_2^5\right) + w_1^3 w_2^2 \left(8\frac{s_1}{s_2} - 10\right) + w_2^3 w_1^2 \left(8\frac{s_2}{s_1} - 10\right) \\
&\qquad + w_1 w_2^4 \left(4\left(\frac{s_1}{s_2}\right)^2 - 5\right) + w_1^4 w_2 \left(4\left(\frac{s_2}{s_1}\right)^2 - 5\right)
\end{aligned}$$

Now because $w_1 \geq \frac{w_2 s_1}{s_2}$, we can bound the last equation. Let $u = \frac{s_1}{s_2}$ (and hence $w_1 \geq u \times w_2$):

$$\begin{aligned}
4\left(\frac{w_1^2}{s_1}+\frac{w_2^2}{s_2}\right)^2 &\left(w_1 s_1^2 + w_2 s_2^2 - (w_1+w_2)\left(\frac{1}{2}\frac{(w_1+w_2)^2}{\frac{w_1^2}{s_1}+\frac{w_2^2}{s_2}}\right)^2\right) \\
&\geq w_2^5\left(3\big(u^5+1\big)+u^3(8u-10)+u^2\left(8\frac{1}{u}-10\right)+u\big(4u^2-5\big)+u^4\left(4\frac{1}{u^2}-5\right)\right) \\
&= w_2^5\big(3u^5+3u^4-6u^3-6u^2+3u+3\big) \\
&= 3w_2^5\left(u-1\right)^2\left(u+1\right)^3.
\end{aligned}$$

Since $w_2 > w_1$, $0 < u < 1$, and this polynomial is strictly positive, hence we have $w_1 s_1^2 + w_2 s_2^2 - 2ws_B^2 > 0$.

Finally, we can conclude that in both cases, $\mathbb{E}(E)((w_1,s_1),(w_2,s_2),\sigma) - \mathbb{E}(E)((w,s_B),(w,s_B),\sigma) > 0$, so there exist a better solution with two chunks of same sizes, hence leading to a contradiction.

We had proven that all chunks have the same size. We use the same line of reasoning to prove that all chunks are executed at a same speed s. If there are two chunks executed at speeds $s_1 < s_2$ (with $w_1 = w_2 = w$), then we have $s_A = s_B$. Considering that $s = s_A$, it is easy to see that $w_1 s_1^2 + w_2 s_2^2 - 2ws_A^2 > 0$ since $w_1 w_2^2 g\left(\frac{s_1}{s_2}\right) + w_1^2 w_2 g\left(\frac{s_2}{s_1}\right) > 0$. Indeed, g is null only in 1, and $s_1 \neq s_2$. We exhibit a solution strictly better, hence showing a contradiction. This concludes the proof. ■

Thanks to this result, we know that the n chunks problem can be rewritten as follows: find s such that

- $\frac{W}{s} + nT_C + \frac{\lambda}{n}\left(\frac{W}{s}+nT_C\right)\left(\frac{W}{\sigma}+nT_C\right) = D$,
- in order to minimize $Ws^2 + nE_C + \frac{\lambda}{n}\left(\frac{W}{s}+nT_C\right)\left(W\sigma^2+nE_C\right)$.

One can see that this reduces to the SINGLECHUNK MULTIPLESPEEDS EXPECTED-DEADLINE task problem where

- $\lambda \leftarrow \frac{\lambda}{n}$,
- $T_C \leftarrow nT_C$,
- $E_C \leftarrow nE_C$,

and allows us to write the problem to solve as a two parameters function:

$$(n,s) \mapsto Ws^2 + nE_C + \frac{\lambda}{n}\left(\frac{W}{s}+nT_C\right)\left(W\left(\frac{\frac{\lambda}{n}W}{\frac{D}{\frac{W}{s}+nT_C}-(1+\lambda T_C)}\right)^2 + nE_C\right), \tag{8.15}$$

which can be minimized numerically.

Hard deadline

In this section, the constraint on the execution time can be written as:

$$\sum_i \left(\frac{w_i}{s_i}+T_C+\frac{w_i}{\sigma_i}+T_C\right) \leq D.$$

Lemma 8.9. *In the* MULTIPLESPEEDS HARD-DEADLINE *model with divisible chunk, the deadline should be tight.*

Proof. This result is obvious with Lemma 8.4: if we have a solution such that the deadline is not tight, if we fix every variable but σ_1 (the re-execution speed of the first task), we can improve the solution with a tight deadline. ■

Lemma 8.10. *In the optimal solution, for all i, j, $\lambda\left(\frac{w_i}{s_i}+T_C\right)\sigma_i^3 = \lambda\left(\frac{w_j}{s_j}+T_C\right)\sigma_j^3$.*

Proof. Consider any solution to our problem. Thanks to Lemma 8.9, we know that the deadline should be tight. Let T_i and T_j be two tasks of re-execution speeds σ_i, σ_j. We show that those speed can be optimally defined such that $\lambda\left(\frac{w_i}{s_i}+T_C\right)\sigma_i^3 = \lambda\left(\frac{w_j}{s_j}+T_C\right)\sigma_j^3$. Let us call $u_i = \lambda\left(\frac{w_i}{s_i}+T_C\right)$ and $u_j = \lambda\left(\frac{w_j}{s_j}+T_C\right)$.

The minimization problem for those speeds can be written as $A_0 + u_i w_i \sigma_i^2 + u_j w_j \sigma_j^2$ under the constraint that $A_1 + \frac{w_i}{\sigma_i} + \frac{w_j}{\sigma_j} = D$ where neither A_0 nor A_1 depends on σ_i, σ_j.

Replacing $\sigma_i = \frac{w_i}{D-A_1-\frac{w_j}{\sigma_j}}$ in the function we need to minimize, we obtain $A_0 + u_i w_i \left(\frac{w_i}{D-A_1-\frac{w_j}{\sigma_j}}\right)^2 + u_j w_j \sigma_j^2$. A simple differentiation gives $-2w_j u_i \frac{w_i^3}{\left(D-A_1-\frac{w_j}{\sigma_j}\right)^3 \sigma_j^2} + 2u_j w_j \sigma_j$. Another differentiation shows the convexity of the function we want to minimize. Hence one can see that the function is minimized when $u_j \sigma_j^3 = u_i \left(\frac{w_i}{D-A_1-\frac{w_j}{\sigma_j}}\right)^3 = u_i \sigma_i^3$. ■

Lemma 8.11. *If we enforce the condition that the execution speeds of the chunks are all equal, and that the re-execution speeds of the chunks are all equal, then all chunks should have same size in the optimal solution.*

Proof. This result is obvious since the problem can be reformulated as the minimization of $\alpha \sum w_i + \beta \sum w_i^2$ where neither α nor β depends on any w_i, under the constraints $\gamma \sum w_i + \zeta \leq D$, and $\sum w_i = W$. It is easy to see the result when there are only two chunks since there is only one variable, and the problem generalizes well in the case of n chunks. ■

We have not been able to prove a stronger result than Lemma 8.11. However we conjecture the following result:

Conjecture 8.1. In the MULTIPLESPEEDS HARD-DEADLINE, in the optimal solution, the re-execution speeds are identical, the deadline is tight. The re-execution speed is equal to $\sigma = \frac{W}{(D-2nT_C)s-W}s$. Furthermore the chunks should have the same size $\frac{W}{n}$ and should be executed at the same speed s.

This conjecture reduces the problem to the SINGLECHUNK MULTIPLESPEEDS problem where

- $\lambda \leftarrow \frac{\lambda}{n}$,
- $T_C \leftarrow nT_C$,
- $E_C \leftarrow nE_C$,

and allows us to write the problem to solve as a two-parameter function:

$$(n, s) \mapsto Ws^2 + nE_C + \frac{\lambda}{n}\left(\frac{W}{s} + nT_C\right)\left(W\left(\frac{W}{(D - 2nT_C)s - W}s\right)^2 + nE_C\right) \tag{8.16}$$

which can be solved numerically.

8.6 Simulations

8.6.1 Simulation settings

We performed a large set of simulations in order to illustrate the differences between all the models studied in this chapter, and to show upon to which extent each additional degree of freedom improves the results, i.e., allowing for multiple speeds instead of a single speed, or for multiple smaller chunks instead of a single large chunk. All these experiments are conducted under both constraint types, expected and hard deadlines.

We envision reasonable settings by varying parameters within the following ranges:

- $\frac{W}{D} \in [0.2, 10]$
- $\frac{T_C}{D} \in [10^{-4}, 10^{-2}]$
- $E_C \in [10^{-3}, 10^3]$
- $\lambda \in [10^{-8}, 1]$.

In addition, we set the deadline to 1. Note that since we study $\frac{W}{D}$ and $\frac{T_C}{D}$ instead of W and T_C, we do not need to study how the variation of the deadline impacts the simulation, this is already taken into account.

We use the Maple software to solve numerically the different minimization problems. Results are showed from two perspectives: on the one hand (Figures 8.1 and 8.2), for a given constraint (HARD-DEADLINE or EXPECTED-DEADLINE), we normalize all variants according to SINGLESPEED SINGLECHUNK, under the considered constraint. For instance, on the plots, the energy consumed by MULTIPLECHUNKS MULTIPLESPEEDS (denoted as MCMS) for HARD-DEADLINE is normalized by the energy consumed by SINGLECHUNK SINGLESPEED (denoted as SCSS) for HARD-DEADLINE, while the energy of MULTIPLECHUNKS SINGLESPEED (denoted as MCSS) for EXPECTED-DEADLINE is normalized by the energy of SINGLECHUNK SINGLESPEED for EXPECTED-DEADLINE.

On the other hand (Figures 8.3 and 8.4), we study the impact of the constraint hardness on the energy consumption. For each solution form (SINGLESPEED or MULTIPLESPEEDS, and SINGLECHUNK or MULTIPLECHUNKS), we plot the ratio energy consumed for EXPECTED-DEADLINE over energy consumed for HARD-DEADLINE.

Note that for each figure, we plot for each function different values that depend on the different values of T_C/D (hence the vertical intervals for points where T_C/D has an impact). In addition, the lower the value of T_C/D, the lower the energy consumption.

8.6.2 Comparison with single speed

At first, we observe that the results are identical for any value of W/D, up to a translation of E_C (see $(W/D = 0.2, E_C = 10^{-3})$ vs. $(W/D = 5, E_C = 1000)$ on Figures 8.1 and 8.2, or see $(W/D = 1, E_C = 10^{-3})$ vs. $(W/D = 5, E_C = 0.1)$ on Figures 8.1 and 8.2, for instance).

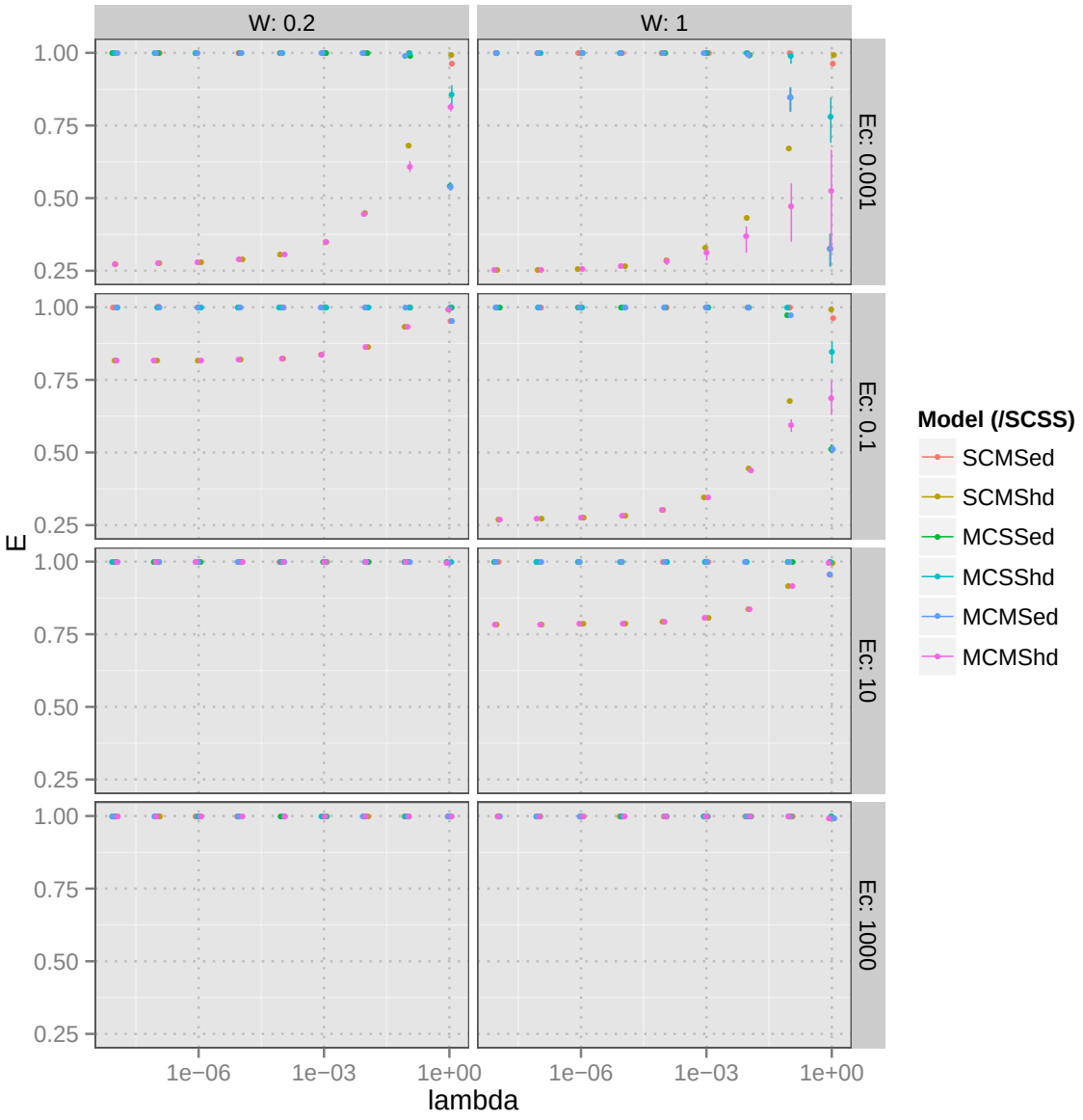


Figure 8.1: Comparison with SINGLECHUNK SINGLESPEED.

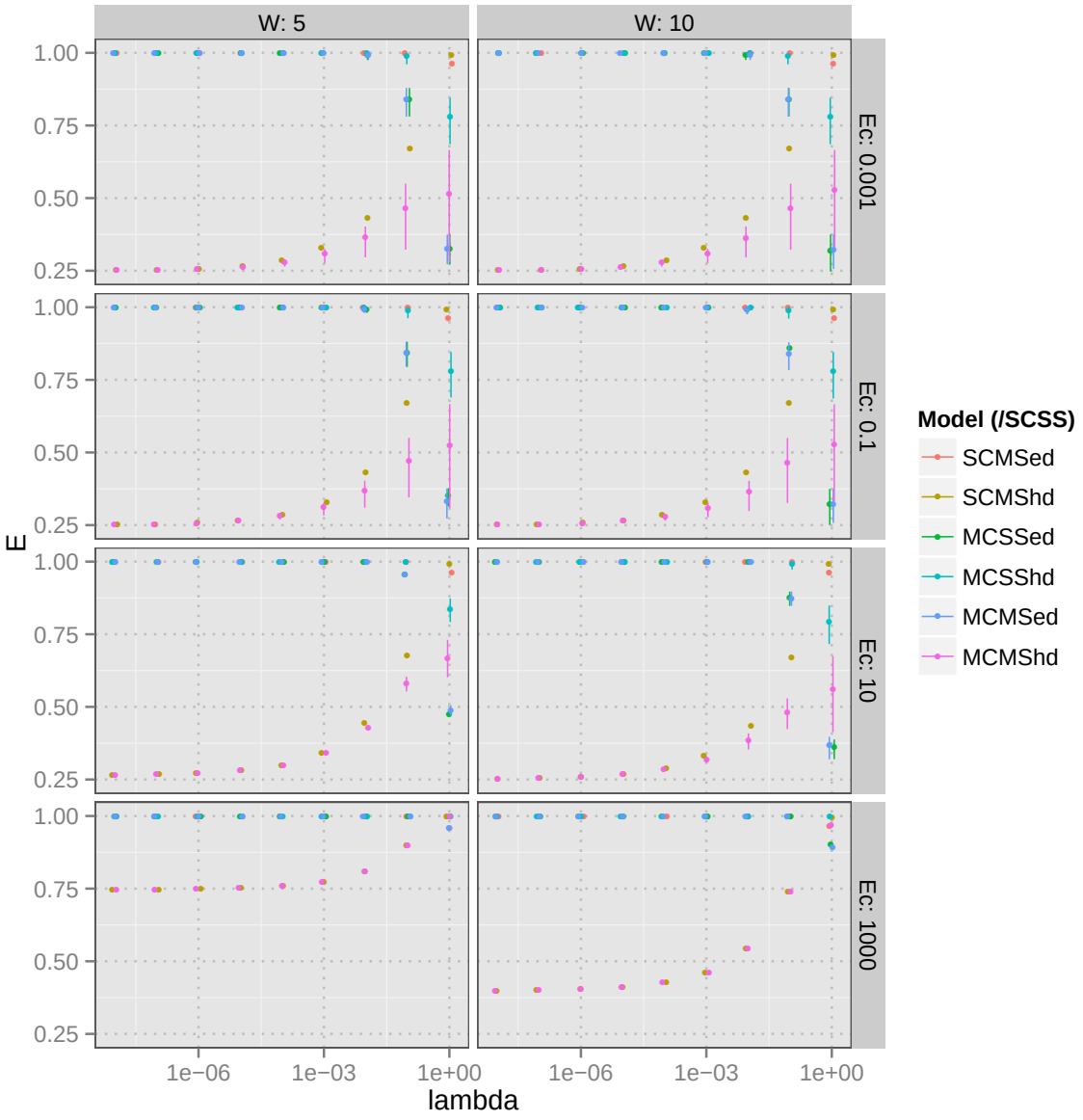


Figure 8.2: Comparison with SINGLECHUNK SINGLESPEED.

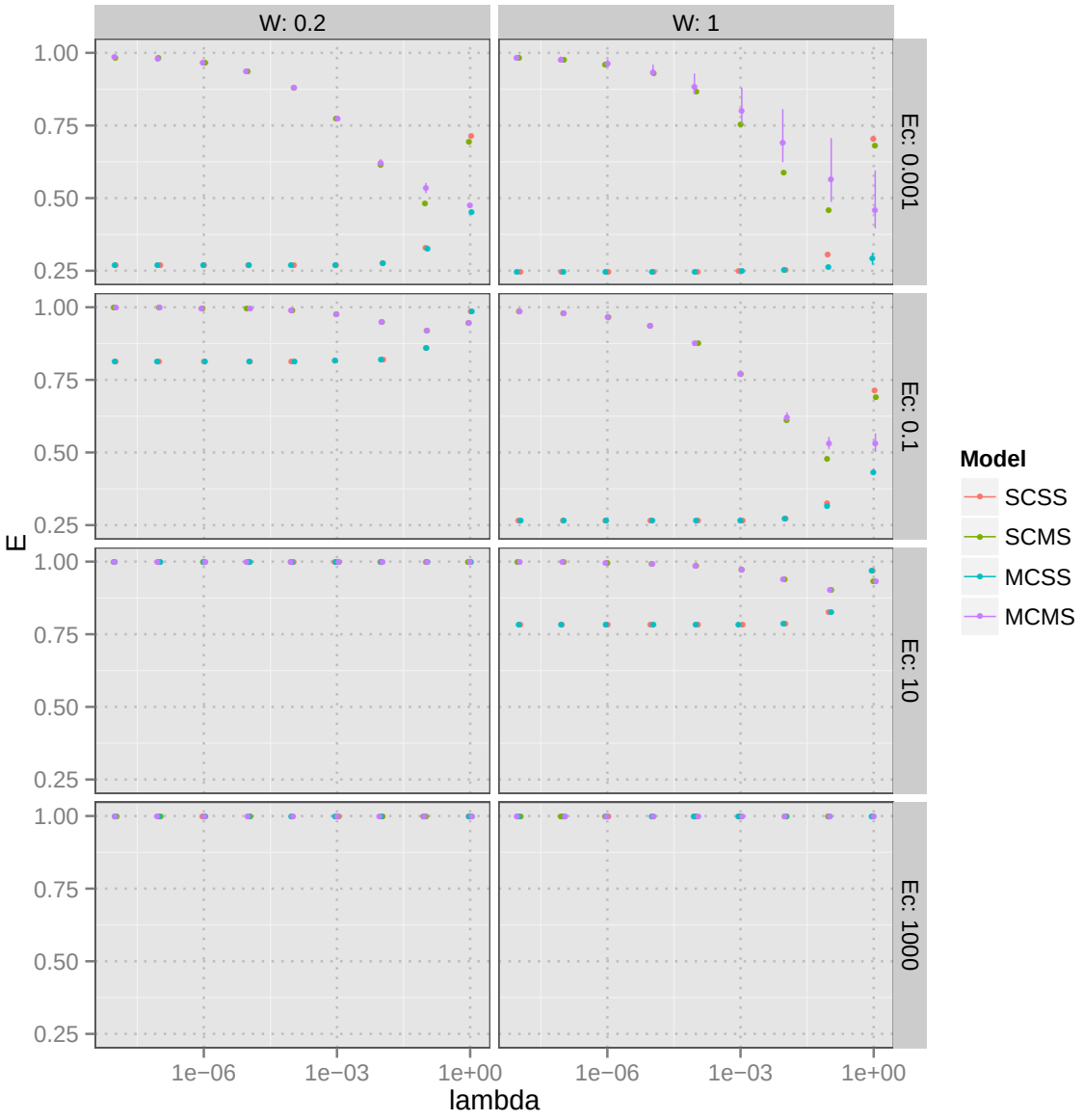


Figure 8.3: Comparison HARD-DEADLINE versus EXPECTED-DEADLINE.

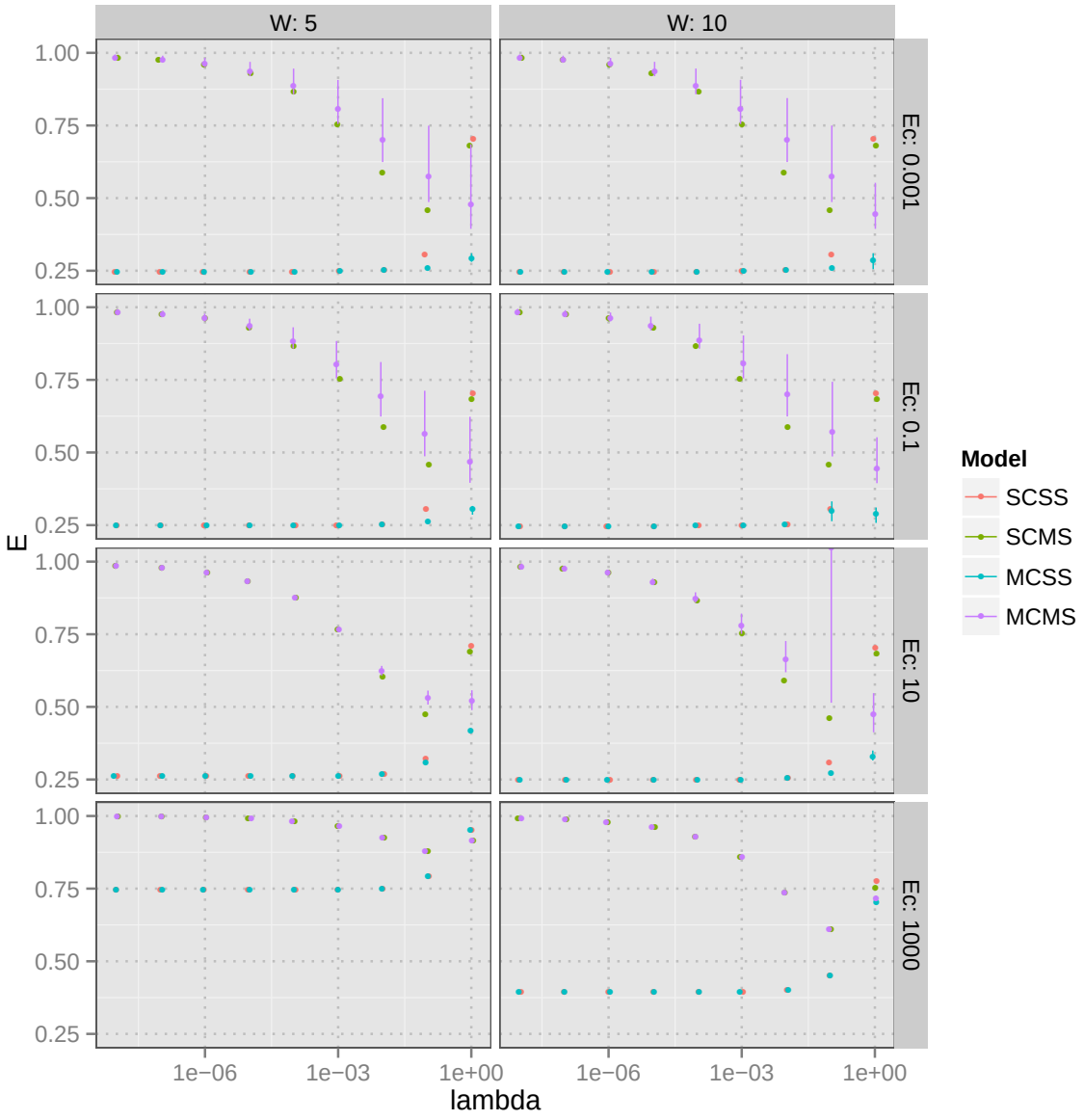


Figure 8.4: Comparison HARD-DEADLINE versus EXPECTED-DEADLINE.

Then the next observation is that for EXPECTED-DEADLINE, with a small λ ($< 10^{-2}$), MULTIPLECHUNKS or MULTIPLESPEEDS models do not improve the energy ratio. This is due to the fact that, in both expressions for energy and for execution time, the re-execution term is negligible relative to the execution one, since it has a weighting factor λ. However, when λ increases, if the energy of a checkpoint is small in front of the total work (which is the general case), we can see a huge improvement (between 25% and 75% energy saving) with MULTIPLECHUNKS.

On the contrary, as expected, for small λ's, re-executing at a different speed has a huge impact for HARD-DEADLINE, where we can gain up to 75% energy when the failure rate is low. We can indeed run at around half speed during the first execution (leading to the $1/2^2 = 25\%$ saving), and at a high speed for the second one, because the very low failure probability avoids the explosion of expected energy consumption. For both MULTIPLECHUNKS and SINGLECHUNK, this saving ratio increases with λ (the energy consumed by the second execution cannot be neglected any more, and both executions need to be more balanced), the latter being more sensitive to λ. But the former is the only configuration where T_C has a significant impact: its performance decreases with T_C; still it remains strictly better than SINGLECHUNK MULTIPLESPEEDS.

8.6.3 Comparison between EXPECTED-DEADLINE and HARD-DEADLINE

As before, the value of W/D does not change the energy ratios up to translations of E_C. As expected, the difference between the EXPECTED-DEADLINE and HARD-DEADLINE models is very important for the SINGLESPEED variant: when the energy of the re-execution is negligible (because of the failure rate parameter), it would be better to spend as little time as possible doing the re-execution in order to have a speed as slow as possible for the first execution, however we are limited in the SINGLESPEED HARD-DEADLINE model by the fact that the re-execution time is fully taken into account (its speed is the same as the first execution, and there is no parameter λ to render it negligible).

Furthermore, when λ is minimum, MULTIPLESPEEDS consumes the same energy for EXPECTED-DEADLINE and for HARD-DEADLINE. Indeed, as expected, the λ in the energy function makes it possible for the re-execution speed to be maximal: it has little impact on the energy, and it is optimal for the execution time; this way we can focus on slowing down the first execution of each chunk. For HARD-DEADLINE, we already run the first execution at half speed, thus we cannot save more energy, even considering EXPECTED-DEADLINE instead. When λ increases, speeds of HARD-DEADLINE cannot be lowered but the expected execution time decreases, making room for a downgrade of the speeds in the EXPECTED-DEADLINE problems.

8.7 Conclusion

In this chapter, we have studied the energy consumption of a divisible computational workload on volatile platforms. In particular, we have studied the expected energy consumption under different deadline constraints: a soft deadline (a deadline for the expected execution time), and a hard deadline (a deadline for the worst case execution time).

We have been able to show mathematically, for all cases but one, that when using the MULTIPLECHUNKS model, then (i) every chunk should be equally sized; (ii) every execution speed should be equal; and (iii) every re-execution speed should also be equal. This problem remains open in the MULTIPLESPEEDS HARD-DEADLINE variant.

Through a set of extensive simulations, we were able to show the following: (i) when the fault parameter λ is small, for EXPECTED-DEADLINE constraints, the SINGLECHUNK SINGLESPEED model

leads to almost optimal energy consumption. This is not true for the HARD-DEADLINE model, which accounts equally for execution and re-execution, thereby leading to higher energy consumption. Therefore, for the HARD-DEADLINE model and for small λ, the model of choice should be the SINGLECHUNK MULTIPLESPEEDS model. When the fault parameter rate λ increases, using a single chunk is no longer energy-efficient, and one should focus on the MULTIPLECHUNKS MULTIPLESPEEDS model for both deadline types.

An interesting direction for future work is to extend this study to the case of an application workflow: instead of dealing with a single divisible task, we would deal with a DAG of tasks, that could be either divisible (checkpoints can take place anytime) or atomic (checkpoints can only take place at the end of the execution of some tasks). Again, we can envision both soft or hard constraints on the execution time, and we can keep the same model with a single re-execution per chunk/task, at the same speed or possibly at a different speed. Deriving complexity results and heuristics to solve this difficult problem is likely to be very challenging, but could have a dramatic impact to reduce the energy consumption of many scientific applications.

Chapter 9

Optimal checkpointing period: Time vs. energy

9.1 Introduction

To conclude this part on energy-efficient and reliable algorithms for Exascale systems, in this last chapter, we study the energy consumption of coordinated checkpointing as seen in Part I. Contrary to Chapters 6, 7 and 8, we do not consider here the speed scaling technique to reduce the energy consumption of the schedules, but we consider the power consumption of the different steps of the coordinated checkpointing technique.

In this chapter, we investigate the trade-offs between execution time and energy consumption for the execution of parallel applications on future Exascale systems. The optimal period $\mathcal{T}_{\text{Time}}^{\text{opt}}$ given by Young's and Daly's formula [124, 35] will minimize the (expected) execution time. However, this period $\mathcal{T}_{\text{Time}}^{\text{opt}}$ will not minimize the energy consumption, mainly because the fraction of power $\mathcal{P}_{\text{Cal}}$ spent when computing (by the CPUs) is not the same as the fraction of power $\mathcal{P}_{\text{I/O}}$ spent when checkpointing. In particular, we revisit the work of Meneses et al. [86] for checkpoint/restart, where formulas are given to compute the time-optimum and energy-optimum periods. However, our model is more precise: (i) we carefully assess the impact of the power consumption required for I/O activity, which is likely to play a key role at the Exascale; (ii) we consider non-blocking checkpointing that can be partially overlapped with computations; (iii) we give a more accurate analysis of the consumed energy.

Main contributions. In this chapter, we provide a refined analytical model to compute both the execution time and the consumed energy with a given checkpoint period. The model handles the case where checkpointing activity can be non-blocking, i.e., partially overlapped with computations. We also provide analytical formulas to approximate the optimal period for time $\mathcal{T}_{\text{Time}}^{\text{opt}}$ as well as the optimal period for energy $\mathcal{T}_{\text{Energy}}^{\text{opt}}$, thereby refining and extending the results by Daly [35] and Meneses et al. [86] to non-blocking checkpoints. Finally, we assess the range of time/energy trade-offs to be made by instantiating the model with a set of realistic scenarios for Exascale systems.

9.2 Model

In this section, we introduce all the model parameters. We start by reminding parameters related to resilience (checkpointing) seen in Part I, with some additional definitions and notations for *blocking checkpointing*, before moving to parameters related to energy consumption. We present in Table 9.1 the main notations used in this chapter.

9.2.1 Checkpointing

We model coordinated checkpointing [29] where checkpoints are taken at regular intervals, after some fixed amount of work-time have been performed. This corresponds to an execution partitioned into periods of duration T. Every period, a checkpoint of length C is taken.

An important question is whether checkpoints are blocking or not. On some architectures, we may have to stop executing the application before writing to the stable storage where the checkpoint data is saved; in that case checkpoint is fully blocking. On other architectures, checkpoint data can be saved on the fly into a local memory before the checkpoint is sent to the stable storage, while computation can resume progress; in that case, checkpoints can be fully overlapped with computations. To deal with all situations, we introduce a slow-down factor ω: during a checkpoint of duration C, the work that is performed is ωC time units. In other words, $(1-\omega)C$ time units are wasted due to checkpoint jitter disrupting the progress of computation. Here, $0 \leq \omega \leq 1$ is an arbitrary parameter. The case $\omega = 0$ corresponds to a fully blocking checkpoint, while $\omega = 1$ corresponds to a checkpoint totally overlapped with computations. All intermediate situations can be represented.

Next we have to account for failures. During t time units of execution, the expectation of the number of failures is $\frac{t}{\mu}$, where μ is the MTBF (Mean Time Between Failures) of the platform. Remember that if the platform is made of N identical resources whose individual mean time between failures is μ_{ind}, then $\mu = \frac{\mu_{\text{ind}}}{N}$. This relation is agnostic of the granularity of the resources, which can be anything from a single CPU to a complex multi-core socket. When a failure strikes, there is a downtime of length D (time to reboot the resource or set up a spare), and then a recovery of length R (time to read the last stored checkpoint). The work executed by the application since the last checkpoint and before the failure needs to be re-executed. Clearly, the shorter the period T, the less work to re-execute, but also the more overhead due to frequent checkpoints in a failure-free execution. The best trade-off when $\omega = 0$ (blocking checkpoint) is achieved for $T = \sqrt{2C(\mu - D - R)}$ (see Chapter 1) or $T = \sqrt{2C(\mu + D + R)} + C$ (Daly's formula [35]). Both formulas are first-order approximations and valid only if all checkpoint parameters C, D and R are small in front of μ (and these formulas collapse if they become negligible). In Section 9.3, we show how to extend these formulas to the case of non-blocking checkpoints (see also [16] for more details).

9.2.2 Energy

To compute the energy consumption of the application, we need to consider the energy consumption of the different phases, and hence the power consumption at each time-step. To this purpose, we define:

- $\mathcal{P}_{\textbf{Static}}$**:** this is the base power consumed when the platform is switched on.

$\mathcal{P}_{\text{Static}}$	Power consumed when platform is switched on
$\mathcal{P}_{\text{Cal}}$	Power consumed due to CPU overhead
$\mathcal{P}_{\text{I/O}}$	Power consumed due to file I/O
$\mathcal{P}_{\text{Down}}$	Power consumed when a machine is down (reboot cost for instance)
$\mathcal{T}_{\text{base}}$	Base execution time (no failure or fault-tolerance overhead)
$\mathcal{T}_{\text{ff}}$	Execution time without failures
$\mathcal{T}_{\text{final}}$	Total execution time
μ	Platform MTBF
ω	Slow down factor for computations during checkpoint

Table 9.1: Table of main notations.

- $\mathcal{P}_{\textbf{Cal}}$**:** when the platform is active, we have to consider the CPU overhead in addition to the static power $\mathcal{P}_{\text{Static}}$.
- $\mathcal{P}_{\textbf{I/O}}$**:** similarly, this is the power overhead due to file I/O. This supplementary power consumption is induced by checkpointing, or when recovering from a failure.
- $\mathcal{P}_{\textbf{Down}}$**:** for coordinated checkpointing, when one processor fails, the rest of the machine stays idle. $\mathcal{P}_{\text{Down}}$ is the power consumption overhead when one machine is down, that may be incurred for instance by rebooting the machine. In general, we let $\mathcal{P}_{\text{Down}} = 0$.

Meneses et al. [86] have a simpler model with two parameters, namely L, the base power (corresponding to $\mathcal{P}_{\text{Static}}$ with our notations), and H, the maximum power (corresponding to $\mathcal{P}_{\text{Static}} + \mathcal{P}_{\text{Cal}}$ with our notations). They use $\mathcal{P}_{\text{I/O}} = \mathcal{P}_{\text{Down}} = 0$.

In Section 9.3, we show how to compute the optimal period that minimizes the energy consumption. In Section 9.4, we instantiate the model with expected values for power consumption of Exascale platforms.

9.3 Optimal checkpointing period

We consider a parallel application whose execution time is $\mathcal{T}_{\text{base}}$ without any overhead due to the resilience method or the occurrence of failures. We compute the expectation $\mathcal{T}_{\text{final}}$ of the total execution time (accounting both for checkpointing and for failures) in Section 9.3.1, and the expectation $\mathcal{E}_{\text{final}}$ of the total energy consumed during this execution of length $\mathcal{T}_{\text{final}}$ in Section 9.3.2. We will compute the optimal period T that minimizes the objective, either $\mathcal{T}_{\text{final}}$ or $\mathcal{E}_{\text{final}}$.

9.3.1 Execution time

The total execution time $\mathcal{T}_{\text{final}}$ of the application depends on two sources of overhead. We first compute $\mathcal{T}_{\text{ff}}$, the time taken by a fault-free execution, thereby accounting only for the overhead due to periodic checkpointing. Then we compute $\mathcal{T}_{\text{fails}}$, the time lost due to failures. Finally, $\mathcal{T}_{\text{final}} = \mathcal{T}_{\text{ff}} + \mathcal{T}_{\text{fails}}$. We detail here both computations:

- The reasoning to derive $\mathcal{T}_{\text{ff}}$ is simple. We need to execute a total amount of time equal to $\mathcal{T}_{\text{base}}$. During each period of length T, there is an amount of time $T - C$ where only computations take place, and an amount of time C units executed during a period of length T is $T - C + \omega C = T - (1 - \omega)C$, and
$$\mathcal{T}_{\text{ff}} = \mathcal{T}_{\text{base}} \frac{T}{T - (1 - \omega)C}.$$
- The reasoning to compute $\mathcal{T}_{\text{fails}}$ is the following. Since the mean time between two failures is μ, the average number of failures during execution is $\frac{\mathcal{T}_{\text{final}}}{\mu}$. For each failure, the time lost is expressed as:
  - $D + R$ for downtime and recovery;
  - a time ωC for the work that was done during the previous checkpoint and that has to be redone because it was not checkpointed (because of the failure);
  - with probability $\frac{T-C}{T}$, the failure happens while we are not checkpointing, and the time lost is on average $A = \frac{T-C}{2}$;

– otherwise, with probability $\frac{C}{T}$, the failure happens while we are checkpointing, and the time lost is on average $B = T - C + \frac{C}{2} = T - \frac{C}{2}$.

The time lost for each failure is

$$D + R + \omega C + \frac{T-C}{T}A + \frac{C}{T}B = D + R + \omega C + \frac{T}{2}.$$

Finally,

$$\mathcal{T}_{\text{fails}} = \frac{\mathcal{T}_{\text{final}}}{\mu}\left(D + R + \omega C + \frac{T}{2}\right).$$

We are now ready to express the total execution time:

$$\begin{aligned}
\mathcal{T}_{\text{final}} &= \mathcal{T}_{\text{ff}} + \mathcal{T}_{\text{fails}} \\
&= \mathcal{T}_{\text{base}}\frac{T}{T-(1-\omega)C} + \frac{\mathcal{T}_{\text{final}}}{\mu}\left(D + R + \omega C + \frac{T}{2}\right) \\
\Leftrightarrow \mathcal{T}_{\text{final}} &= \frac{T}{(T-(1-\omega)C)\left(1 - \frac{D+R+\omega C+T/2}{\mu}\right)}\mathcal{T}_{\text{base}} \\
&= \frac{T}{(T-a)\left(b - \frac{T}{2\mu}\right)}\mathcal{T}_{\text{base}},
\end{aligned}$$

where $a = (1-\omega)C$ and $b = 1 - \frac{D+R+\omega C}{\mu}$.

This equation is minimized for

$$\mathcal{T}_{\text{Time}}^{\text{opt}} = \sqrt{2(1-\omega)C(\mu - (D + R + \omega C))}. \tag{9.1}$$

As a sanity check, when $\omega = 0$, we obtain the expression from Chapter 1 for RFO. In the following, we let ALGOT be the checkpointing strategy that checkpoints with period $\mathcal{T}_{\text{Time}}^{\text{opt}}$.

9.3.2 Energy consumption

In order to compute the total energy consumption of the execution, we consider the different phases during which the different powers introduced in Section 9.2.2 are used:

- First, we consume $\mathcal{P}_{\text{Static}}$ during each time-step of the execution. Indeed, even when a node fails and is shutdown, we still pay for the power of all the other nodes, for the cooling system, etc. The corresponding energy cost is $\mathcal{T}_{\text{final}}\mathcal{P}_{\text{Static}}$.

- Next, let $\mathcal{T}_{\text{Cal}}$ be the time during which the CPU is used, inducing a power overhead $\mathcal{P}_{\text{Cal}}$. $\mathcal{T}_{\text{Cal}}$ includes the base time $\mathcal{T}_{\text{base}}$, and $\mathcal{T}_{\text{re-exec}}$, the time to compute the work that must be re-executed after each failure (which we multiply by the number of failures $\mathcal{T}_{\text{final}}/\mu$):

  – with probability $\frac{T-C}{T}$, the failure does not happen during a checkpoint, and the work to re-execute lasts $A = \omega C + \frac{T-C}{2}$;
  – with probability $\frac{C}{T}$, the failure happens during the execution of a checkpoint, and the work to re-execute lasts $B = \omega C + T - C + \frac{\omega C}{2}$.

We derive $\mathcal{T}_{\text{re-exec}} = \frac{T-C}{T}A + \frac{C}{T}B$, hence

$$\mathcal{T}_{\text{re-exec}} = \omega C + \frac{T^2 - C^2}{2T} + \frac{\omega C^2}{2T}.$$

Finally, we have:

$$\mathcal{T}_{\text{Cal}} = \mathcal{T}_{\text{base}} + \frac{\mathcal{T}_{\text{final}}}{\mu}\left(\omega C + \frac{T^2 - C^2}{2T} + \frac{\omega C^2}{2T}\right).$$

The corresponding energy consumption is $\mathcal{T}_{\text{Cal}}\mathcal{P}_{\text{Cal}}$.

- Let $\mathcal{T}_{\text{I/O}}$ be the time during which the I/O system is used, inducing a power overhead $\mathcal{P}_{\text{I/O}}$. This time corresponds to checkpointing and recovery from failures.
  - The total number of checkpoints that are taken in a fault-free execution is equal to the number of periods, $\frac{\mathcal{T}_{\text{base}}}{T-(1-\omega)C}$, and the time taken by checkpoints is therefore $\frac{\mathcal{T}_{\text{base}}C}{T-(1-\omega)C}$.
  - For each failure, there is an additional overhead:
    1. the system needs to recover, which lasts R time-steps;
    2. with probability $\frac{T-C}{T}$, the failure does not happen during a checkpoint, and there is no additional I/O overhead;
    3. however, with probability $\frac{C}{T}$, the failure happens during a checkpoint, and the I/O time wasted is (in average) $\frac{C}{2}$.

  Altogether, we obtain

$$\mathcal{T}_{\text{I/O}} = \frac{\mathcal{T}_{\text{base}}C}{T-(1-\omega)C} + \frac{\mathcal{T}_{\text{final}}}{\mu}\left(R + \frac{C^2}{2T}\right).$$

  The corresponding energy consumption is $\mathcal{T}_{\text{I/O}}\mathcal{P}_{\text{I/O}}$.

- Finally, let $\mathcal{T}_{\text{Down}}$ be the total down time, incurring a power overhead $\mathcal{P}_{\text{Down}}$. We have

$$\mathcal{T}_{\text{Down}} = \frac{\mathcal{T}_{\text{final}}}{\mu}D,$$

  and the corresponding energy cost is $\mathcal{T}_{\text{Down}}\mathcal{P}_{\text{Down}}$. This term is only included for full generality, as we expect to have $\mathcal{P}_{\text{Down}} = 0$ in most scenarios.

The final expression for the total energy consumed is

$$\begin{aligned}
\mathcal{E}_{\text{final}} &= \mathcal{T}_{\text{Cal}}\mathcal{P}_{\text{Cal}} + \mathcal{T}_{\text{I/O}}\mathcal{P}_{\text{I/O}} + \mathcal{T}_{\text{Down}}\mathcal{P}_{\text{Down}} + \mathcal{T}_{\text{final}}\mathcal{P}_{\text{Static}} \\
&= \left(\mathcal{T}_{\text{base}} + \frac{\mathcal{T}_{\text{final}}}{\mu}\left(\omega C + \frac{T^2 - C^2}{2T} + \frac{\omega C^2}{2T}\right)\right)\mathcal{P}_{\text{Cal}} \\
&\quad + \left(\frac{\mathcal{T}_{\text{final}}}{\mu}\left(R + \frac{C^2}{2T}\right) + C\frac{\mathcal{T}_{\text{base}}}{T-(1-\omega)C}\right)\mathcal{P}_{\text{I/O}} + \frac{\mathcal{T}_{\text{final}}}{\mu}D\mathcal{P}_{\text{Down}} + \mathcal{T}_{\text{final}}\mathcal{P}_{\text{Static}}.
\end{aligned}$$

It is important to understand that $\mathcal{T}_{\text{final}} \neq \mathcal{T}_{\text{Cal}} + \mathcal{T}_{\text{I/O}} + \mathcal{T}_{\text{Down}}$, unless $\omega = 0$. Indeed, CPU and I/O activities are overlapped (and both consumed) when checkpointing. To ease the derivation of the optimal period that minimizes $\mathcal{E}_{\text{final}}$, we introduce some notations and let $\mathcal{P}_{\text{Cal}} = \alpha\mathcal{P}_{\text{Static}}$, $\mathcal{P}_{\text{I/O}} = \beta\mathcal{P}_{\text{Static}}$, and $\mathcal{P}_{\text{Down}} = \gamma\mathcal{P}_{\text{Static}}$. Re-using parameters $a = (1-\omega)C$ and $b = 1 - \frac{D+R+\omega C}{\mu}$ from Section 9.3.1, we obtain:

$$\frac{\mathcal{T}'_{\text{final}}}{\mathcal{T}_{\text{base}}} = \frac{-ab + \frac{T^2}{2\mu}}{(T-a)^2\left(b-\frac{T}{2\mu}\right)^2}, \quad \text{and}$$

$$\begin{aligned}\frac{\mathcal{E}'_{\text{final}}}{\mathcal{P}_{\text{Static}}} =& \frac{\mathcal{T}'_{\text{final}}}{\mu}\left(\alpha\omega C+\beta R+\gamma D+\frac{\alpha T}{2}-\frac{\alpha(1-\omega)C^2}{2T}+\frac{\beta C^2}{2T}+\mu\right)\\ &+\frac{\mathcal{T}_{\text{final}}}{2\mu}\left(\alpha+\frac{\alpha(1-\omega)C^2}{T^2}-\frac{\beta C^2}{T^2}\right)-\frac{\beta C\mathcal{T}_{\text{base}}}{(T-(1-\omega)C)^2}.\end{aligned}$$

Then, letting $K = \frac{(T-a)^2\left(b-\frac{T}{2\mu}\right)^2}{\mathcal{P}_{\text{Static}}\mathcal{T}_{\text{base}}}$, we have:

$$\begin{aligned}K\mathcal{E}'_{\text{final}} =& \frac{-ab+\frac{T^2}{2\mu}}{\mu}\left((\alpha\omega C+\beta R+\gamma D+\mu)+\frac{\alpha T}{2}+\frac{\alpha(1-\omega)C^2}{2T}+\frac{\beta C^2}{2T}\right)\\ &+\frac{(T-a)(b-\frac{T}{2\mu})}{2\mu}\left(\alpha+\frac{\alpha(1-\omega)C^2-\beta C^2}{T}\right)-\beta C\left(b-\frac{T}{2\mu}\right)^2\\ =& T^3\left(\frac{1}{4\mu}-\frac{1}{4\mu}\right)+T^2\left(\frac{\alpha\omega C+\beta R+\gamma D}{2\mu^2}+\frac{b+\frac{a}{2\mu}}{2\mu}-\frac{\beta C}{4\mu^2}+\frac{1}{2\mu}\right)\\ &+T\left(-\frac{ab}{2\mu}-\frac{ab}{2\mu}+\frac{\beta Cb}{\mu}-2\frac{(\alpha(1-\omega)-\beta)C^2}{4\mu^2}\right)-\beta Cb^2\\ &-\frac{ab\,(\alpha\omega C+\beta R+\gamma D+\mu)}{\mu}-\left(\frac{b}{2\mu}-\frac{a}{4\mu^2}\right)(\alpha(1-\omega)-\beta)C^2\\ &+\frac{1}{T}\left((\alpha(1-\omega)-\beta)\frac{C}{2\mu}-(\alpha(1-\omega)-\beta)\frac{C}{2\mu}\right)\\ =& T^2\left(\frac{\alpha\omega C+\beta R+\gamma D}{2\mu^2}+\frac{b}{2\mu}+\frac{a-\beta C}{4\mu^2}+\frac{1}{2\mu}\right)\\ &+T\left(\frac{(\beta C-a)b}{\mu}-2\frac{(\alpha(1-\omega)-\beta)C^2}{4\mu^2}\right)\\ &-\frac{ab\,(\alpha\omega C+\beta R+\gamma D+\mu)}{\mu}-\beta Cb^2\\ &+\left(\frac{b}{2\mu}+\frac{a}{4\mu^2}\right)(\alpha(1-\omega)-\beta)C^2\,.\end{aligned}$$

Let $\mathcal{T}^{\text{opt}}_{\text{Energy}}$ be the only positive root of this quadratic polynomial in T: $\mathcal{T}^{\text{opt}}_{\text{Energy}}$ is the value that minimizes $\mathcal{E}_{\text{final}}$. In the following, we let ALGOE be the checkpointing strategy that checkpoints with period $\mathcal{T}^{\text{opt}}_{\text{Energy}}$.

As a side note, let us emphasize the differences with the approach of Meneses et al. [86] when restricting to the case $\omega = 0$ (because they only consider the blocking variant). For each failure, they consider that:

- energy lost due to re-execution is $\frac{T-2C}{2}\mathcal{P}_{\text{Cal}}$, while we have $\left(\frac{T-C}{T}\left(\frac{T-C}{2}\right)+\frac{C}{T}\left(T-C\right)\right)\mathcal{P}_{\text{Cal}} = \frac{T^2-C^2}{2T}\mathcal{P}_{\text{Cal}}$;

- energy lost due to I/O is $C\mathcal{P}_{\text{I/O}}$, while we have $\frac{C^2}{2T}\mathcal{P}_{\text{I/O}}$.

Theses differences come from our more detailed analysis of the impact of the failure location, which can strike either during the computation phase, or during the checkpointing phase, of the whole period.

9.4 Experiments

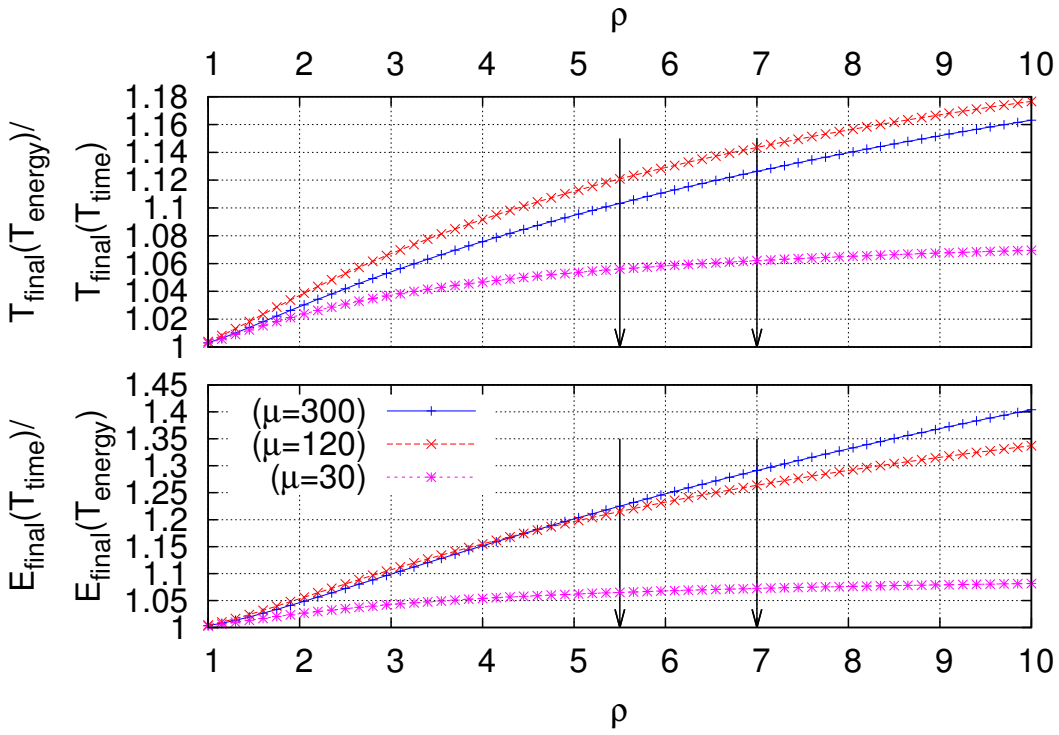


Figure 9.1: Time and energy ratios as a function of ρ, with $C = R = 10$ min, $D = 1$ min, $\gamma = 0$, $\omega = 1/2$, and various values for μ.

In this section, we instantiate the previous model with scenarios taken from current projections for Exascale platforms [39, 108, 111, 43]. We choose realistic values for all model parameters: this includes all types of power consumption ($\mathcal{P}_{\text{Static}}$, $\mathcal{P}_{\text{Cal}}$, $\mathcal{P}_{\text{I/O}}$ and $\mathcal{P}_{\text{Down}}$), all checkpoint parameters (C, R, D and ω), and the platform MTBF μ. We start with a word of caution: our choices for these parameters may be somewhat arbitrary, and do not cover the whole range of scenarios that can be investigated. However, a key feature of our model is its robustness: as long as μ is reasonably large in front of checkpoint times, the model is able to accurately predict the best period for execution time and for energy consumption.

The power consumption of an Exascale machine is capped to 20 Mega-watts. With 10^6 nodes, this represents a nominal power of 20 watts per node. Let us express all power values in watts. A reasonable scenario is to assume that half this power is used for operating the platform, hence to let $\mathcal{P}_{\text{Static}} = 10$. The overhead due to computing would represent the other half, hence $\mathcal{P}_{\text{Cal}} = 10$. As for communications and I/Os, which are expected to cost an order of magnitude more than computing [111], we take an overhead of 100, hence $\mathcal{P}_{\text{I/O}} = 100$. A key parameter for the experimental study is the ratio

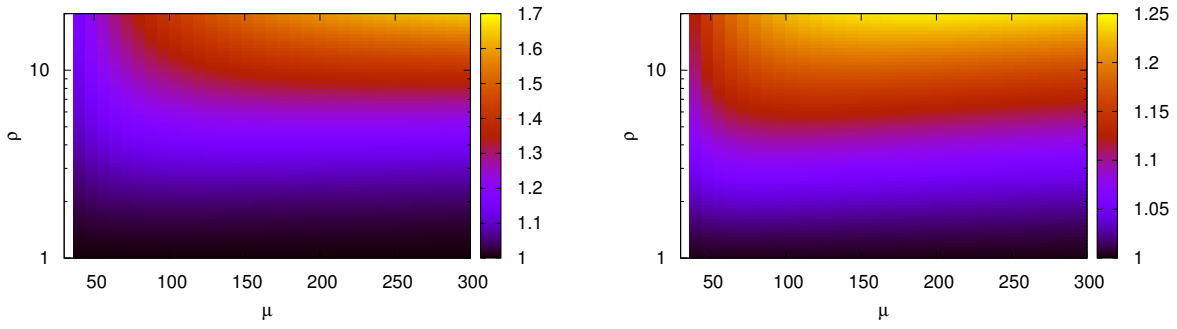
$$\rho = \frac{\mathcal{P}_{\text{Static}} + \mathcal{P}_{\text{I/O}}}{\mathcal{P}_{\text{Static}} + \mathcal{P}_{\text{Cal}}} = \frac{1+\beta}{1+\alpha}. \tag{9.2}$$

With our values, we get $\rho = 5.5$. Note that if we used $\mathcal{P}_{\text{Static}} = 5$ and kept the same overheads 10 and 100 for computing and I/O respectively, we would get $\mathcal{P}_{\text{Cal}} = 10$, $\mathcal{P}_{\text{I/O}} = 100$, and $\rho = 7$. These two representative values of ρ ($\rho = 5.5$ and $\rho = 7$) are emphasized by vertical arrows in the plots below on Figure 9.1. As for $\mathcal{P}_{\text{Down}}$, the power during downtime, we use $\mathcal{P}_{\text{Down}} = 0$, meaning that during downtime we only account for the static power $\mathcal{P}_{\text{Static}}$ of the processors that are idle.

The Jaguar platform, with $N = 45,208$ processors, is reported to have experienced about one fault per day [128], which leads to an individual (processor) MTBF μ_{ind} equal to $\frac{45,208}{365} \approx 125$ years. Therefore, we set the individual (processor) MTBF to $\mu_{\text{ind}} = 125$ years. Letting the total number of processors N vary from $N = 219,150$ to $N = 2,191,500$ (future Exascale platforms), the platform MTBF μ varies from $\mu = 300$ min (5 hours) down to $\mu = 30$ min. The experiments use resilience parameters that are representative of current and forthcoming large-scale platforms [43, 25]. We take $C = R = 10$ min, $D = 1$ min, and $\omega = 1/2$.

On Figures 9.1 and 9.2, we evaluate the impact of the ratio ρ (see Equation (9.2)) on the gain in energy and loss in time of ALGOE with respect to ALGOT. The general trend is that using ALGOE can lead to significant gains in energy at the price of a small increase in execution time.

We then study in Figure 9.3 the scalability of the approach on forthcoming platforms. We set the duration of the complete checkpoint and rollback (C and R, respectively) to 1 minute, independently of the number of processors, and we let the downtime D equal to 0.1 minutes. It is reasonable to consider that checkpoint storage time will not increase with the number of nodes in the future, but on the contrary will remain constant. Indeed, system designers are studying a couple of alternative approaches. One consists of providing each computing node with local storage capability, ensuring through hardware mechanisms that this storage will remain available during a failure of the node. Another approach consists of using the memory of the other processors to store the checkpoint, pairing nodes as "buddies", thus allowing to take advantage of the high bandwidth capability of the high speed network to design a scalable checkpoint storage mechanism [129, 91, 40, 102].



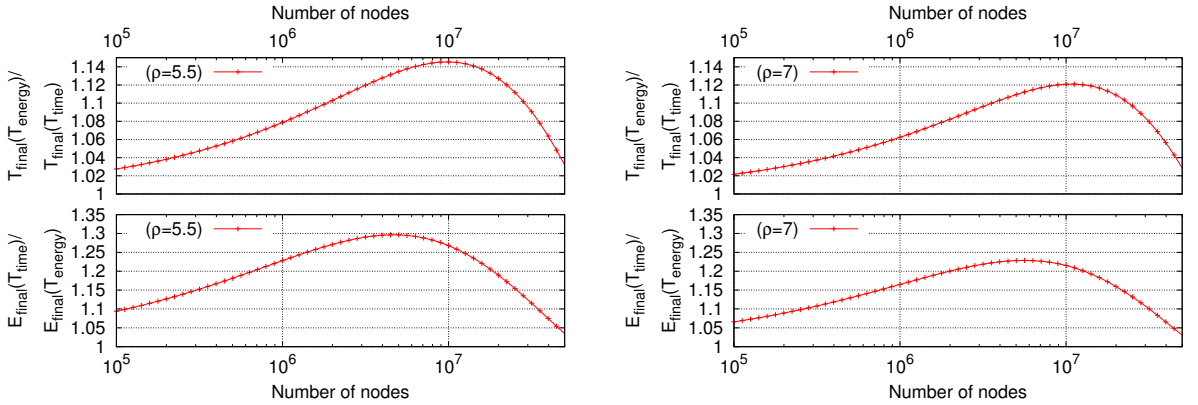
(a) Energy ratio of ALGOT over ALGOE

(b) Execution time ratio of ALGOE over ALGOT

Figure 9.2: Ratios of the different strategies with $C = R = 10$ min, $D = 1$ min, $\gamma = 0$, $\omega = 1/2$ as a function of μ and ρ.

The MTBF for 10^6 nodes is set to 2 hours, and this value scales linearly with the number of com-

ponents. Given these parameters, Figures 9.3a and 9.3b shows (i) the execution time ratio of ALGOE over ALGOT, and (ii) the energy consumption ratio of ALGOT over ALGOE, both as a function of the number of nodes. Figures 9.3a and 9.3b confirm the important gain in energy that can be achieved, namely up to 30% for a time overhead of only 12%. When the number of nodes gets very high (up to 10^8), then we observe that both energy and time ratios converge to 1. Indeed, when C becomes of the order of magnitude of the MTBF, then both periods $\mathcal{T}_{\text{Time}}^{\text{opt}}$ and $\mathcal{T}_{\text{Energy}}^{\text{opt}}$ become close to C to account for the higher failure rate.



(a) Time and energy ratios, as a function of the number of nodes, when $\rho = 5.5$

(b) Time and energy ratios, as a function of the number of nodes, when $\rho = 7$

Figure 9.3: Ratios of total energy and time for the two period strategies, as a function of the number of nodes, with $\mu = 120$ min for 10^6 nodes, $C = R = 1$ min, $D = 0.1$ min, $\gamma = 0$, $\omega = 1/2$.

9.5 Conclusion

In this chapter, we have provided a detailed analysis to compute the optimal checkpointing period, when the checkpointing activity can be partially overlapped with computations. We have considered two distinct objectives: either the goal is to minimize the total execution time, or it is to minimize the total energy consumption. Because of the different power consumption overheads due to computations and I/Os, we obtain different optimal periods.

We have instantiated the formulas with values derived from current and future Exascale platforms, and we have studied the impact of the power overhead due to I/O activity on the gains in time and energy. With current values, we can save more than 20% of energy with an MTBF of 300 min, at the price of an increase of 10% in the execution time. The maximum gains are expected for a platform with between 10^6 and 10^7 processors (up to 30% energy savings).

Our analytical model is quite flexible and can easily be instantiated to investigate scenarios that involve a variety of resilience and power consumption parameters.

Conclusion

In this thesis, we have studied two of the main challenges for Exascale platforms: reliability and energy performance. For the checkpointing techniques introduced in Part I, we proposed optimal algorithms, while we proposed NP-completeness proofs and approximation results for the scheduling problems of Part II. On the practical side, we implemented the algorithms presented in Part I to verify their impact on traces of actual faults and on simulated faults. We designed many efficient polynomial-time heuristics on general problems that were shown NP-complete beforehand. Our main contributions are stated in the following paragraphs.

Summary

Checkpointing algorithms and fault prediction

In this first study, we assessed the impact of fault prediction on periodic checkpointing. We established analytical conditions stating whether a fault prediction should be taken into account or not. More importantly, we proved that the optimal approach is to never trust the predictor in the beginning of a regular period, and to always trust it in the end of the period; the cross-over point $\frac{C_p}{p}$ depends on the time needed to take a proactive checkpoint and on the precision of the predictor.

We conducted simulations involving synthetic failure traces following either an Exponential distribution law or a Weibull one. We also used log-based failure traces. Through this extensive experiment setting, we established the accuracy of the model, of its analysis, and of the predicted period (in the presence of a fault predictor). The simulations also show that even a not-so-good fault predictor can lead to quite a significant decrease in the application execution time.

Checkpointing strategies with prediction windows

This chapter was a natural extension of the previous one. We added an interval of time to the definition of the predictor, during which the prediction is valid. We proposed a new approach based upon two periodic modes: a regular mode outside prediction windows, and a proactive mode inside prediction windows, whenever the size of these windows is large enough. We were able to fully solve this problem with results corroborated by a full set of simulations.

On the combination of silent error detection and checkpointing

In this chapter, we focused on silent data corruption errors. Contrary to fail-stop failures, such latent errors cannot be detected immediately, and a mechanism to detect them must be provided. We provided a general framework to solve, independently of the verification mechanism, the problem of minimizing the execution time of a schedule. We instantiated the model using realistic scenarios and application/architecture parameters.

Energy-aware scheduling under reliability and makespan constraints

In this chapter, we aimed at minimizing the energy consumption of a DAG while enforcing two constraints: a prescribed bound on the execution time (or makespan), and a reliability threshold. We used the re-execution technique coupled with the DVFS technique. We assessed the intractability of this tri-criteria problem, even when the mapping of tasks to processors is already known. In addition, we provided several complexity results for particular instances. Then, based on those results, we designed and evaluated some polynomial-time heuristics for the TRI-CRIT-CONT problem, based on the failure probability, the task weights, and the processor speeds. After running several heuristics on a wide class of problem instances, we identified two complementary heuristics that are able to produce good results on most instances.

Approximation algorithms for energy, reliability and makespan optimization problems

This chapter is a natural extension of the previous one as we not only considered the re-execution technique for reliability, but also the replication technique. This chapter focused on two different applications: linear chains of tasks and a set of independent tasks. For both problems, we presented efficient approximation algorithms. For linear chains, we designed a fully polynomial-time approximation scheme. However, we showed that there does not exist any constant factor approximation algorithm for independent tasks, unless P=NP, and we were able in this case to propose an approximation algorithm with a relaxation on the makespan constraint.

Energy-aware checkpointing of divisible tasks with soft or hard deadlines

This chapter is different from the two previous chapters as it considered divisible tasks. For divisible tasks, the re-execution and replication techniques do not make as much sense as for atomic tasks, so we used checkpoints to resolve the reliability issue. We also considered different kinds of deadlines: a soft deadline and a hard deadline. For each problem instance, we proposed either an exact solution, or a function that could be numerically optimized. Finally, we used thorough simulations to show the efficiency of the various solutions depending on the fault parameter.

Optimal checkpointing period: time vs. energy

In the last chapter, we considered parallel scientific applications using non-blocking and periodic coordinated checkpointing to enforce resilience. We provided a detailed analysis to compute the optimal checkpointing period when the checkpointing activity can be partially overlapped with computations. We considered two distinct objectives: either the goal is to minimize the total execution time, or to minimize the total energy consumption. Because of the different power consumption overheads due to computations and I/Os, we obtained different optimal periods. We instantiated the formulas with values derived from current and future Exascale platforms, and we studied the impact of the power overhead due to I/O activity on the gains in time and energy. With current values, we can save more than 20% of energy with an MTBF of 300 min, at the price of an increase of 10% in the execution time. The maximum gains are expected for a platform with between 10^6 and 10^7 processors (up to 30% energy savings).

Other work

In this document, we have focused on two of the main challenges for Exascale systems: reliability and energy. However, during this thesis, we have also worked on some other challenges for Exascale systems. We provide below a description of the studies that we have been involved with.

Implementing a systolic algorithm for QR factorization on multicore clusters with PaRSEC [C8]

In this work, we consider the problem of algorithms for Exascale architectures: we introduce a new systolic algorithm for QR factorization and its implementation on a supercomputing cluster of multicore nodes. The algorithm targets a virtual 3D-array and requires only local communications. The implementation of the algorithm uses threads at the node level, and MPI for inter-node communications. The complexity of the implementation is addressed with the PaRSEC software, which takes as input a parameterized dependence graph, which is derived from the algorithm, and which only requires the user to decide, at the high-level, on the allocation of tasks to nodes. We show that the new algorithm exhibits competitive performance with state-of-the-art QR routines on a supercomputer called Kraken, which shows that high-level programming environments, such as PaRSEC, provide a viable alternative to enhance the production of quality software on complex and hierarchical architectures.

Scheduling the I/O of HPC applications under congestion [RR11]

In this work, we consider the I/O problem. A significant percentage of the computing capacity of large-scale platforms is wasted due to interferences incurred by multiple applications that concurrently access a shared parallel file system. One solution to handling I/O bursts in large-scale HPC systems is to absorb them at an intermediate storage layer consisting of burst buffers. However, our analysis of the Argonne's Mira system shows that burst buffers cannot prevent congestion at all time. As a consequence, I/O performance is dramatically degraded, showing in some cases a decrease in I/O throughput of 67%. In this work, we analyze the effects of interference on application I/O bandwidth, and propose several scheduling techniques to mitigate congestion. We show through extensive experiments that our global I/O scheduler is able to reduce the effects of congestion, even on systems where burst buffers are used, and can increase the overall system throughput up to 56%. We also show that it outperforms current Mira I/O schedulers.

Co-scheduling algorithms for high-throughput workload execution [RR10]

In this work, we investigate co-scheduling algorithms for processing a set of parallel applications. Instead of executing each application one by one, using a maximum degree of parallelism for each of them, we aim at scheduling concurrently several applications. We partition the original application set into a series of packs, which are executed one by one. A pack comprises several applications, each of them with an assigned number of processors, with the constraint that the total number of processors assigned within a pack does not exceed the maximum number of available processors. The objective is to determine a partition into packs, and an assignment of processors to applications that minimizes the sum of the execution times of the packs. We thoroughly study the complexity of this optimization problem, and propose several heuristics that exhibit very good performance on a variety of workloads, whose application execution times model profiles of parallel scientific codes. We show that co-scheduling leads to faster workload completion time (40% improvement on average over traditional scheduling) and to

faster response times (50% improvement). Hence co-scheduling increases system throughput and saves energy, leading to significant benefits from both the user and system perspectives.

Power-aware replica placement in tree networks with multiple servers per client [C6]

Finally, we revisit the well-studied problem of replica placement in tree networks. Rather than minimizing the number of servers needed to serve all client requests, we aim at minimizing the total power consumed by these servers. In addition, we use the most general (and powerful) server assignment policy, where the requests of a client can be served by multiple servers located in the (unique) path from this client to the root of the tree. We consider multi-modal servers that can operate at a set of discrete speeds, using the dynamic voltage and frequency scaling (DVFS) technique. The optimization problem is to determine an optimal location of the servers in the tree, as well as the speed at which each server is operated. A major result is the NP-completeness of this problem, to be contrasted with the minimization of the number of servers, which has polynomial complexity. Another important contribution is the formulation of a Mixed Integer Linear Program (MILP) for the problem, together with the design of several polynomial-time heuristics. We assess the efficiency of these heuristics by simulation. For mid-size instances (up to 30 nodes in the tree), we evaluate their absolute performance in comparison with the optimal solution (obtained via the MILP). The most efficient heuristics provide satisfactory results, within 20% of the optimal solution.

Perspectives

Throughout the thesis, we pointed out at the end of each chapter some future work that remains to be done. Those, along with the following two main directions for each part form the immediate research that could follow the thesis in the short term. Two more topics, with greater ambitions, follow and complement the work done in this thesis, in a the long term. We label these two with General perspectives

Reliability via periodic checkpointing

In the first part of the thesis, we have discussed different types of faults: fail-stop, and silent errors. Fail-stop failures are such that when they occur, the process stops and the machine needs to be restarted after a down time before being able to do additional work. On the other hand, silent errors are detected only after a latency. The first interesting direction would be to create a checkpointing strategy that would take into account these two kinds of faults depending on their respective MTBF. Furthermore, it could be coupled with some prediction mechanism that may probably not be able to predict silent errors.

Reliable and energy-aware schedules

Most of the work presented in the second part of the thesis considers that a processor can choose any speed in an interval $[f_{\min}, f_{\max}]$. The first direction is to extend the work done on a discrete set of speeds. More specifically, in Chapter 6, we have designed heuristics for the TRI-CRIT-CONT model, but we could easily adapt them to a discrete model. For a solution given by a heuristic for TRI-CRIT-CONT, if a task should be executed at the continuous speed f, then we would execute it at the closest discrete speed above f, while matching the execution time and reliability constraint for this task. There remains to quantify the performance loss incurred by the latter constraints.

General perspective: uncoordinated checkpointing

In this thesis, the primary focus for reliability was periodic coordinated checkpointing. However, coordinated checkpointing has two major issues:

- global restart wastes energy since all processes are forced to rollback even in the case of a single failure [42];
- checkpoint coordination may slow down the application execution because of the volume of I/O data and congestion on I/O resources [95].

Because of these issues, it is not sure that coordinated checkpoint is viable at scale. An alternative approach is the uncoordinated checkpointing method [119, 21, 57]. Bosilca et al. [16] presented hierarchical checkpointing. In their model, all processors do not checkpoint at the same time: they checkpoint as a coordinated group. When a group fails, then the group needs to rollback. This leaves out the two major issues of coordinated checkpointing. Hierarchical checkpointing also uses message logging to get rid of the "domino effect": if no set of checkpoints forms a consistent global state, the application has to be restarted from the beginning in the event of a failure. It makes recovery cost unacceptable and garbage collection complex to implement [42]. Many papers have tried to cope with this issue [57]. Few authors have studied the impact of this technique on the energy consumption of future platforms; Diouri et al [37] have measured the different costs incurred by the various steps of uncoordinated checkpointing, but have not proposed any solution to minimize the energy consumption of this problem.

An interesting future direction for this thesis would be to study the impact of other fault-tolerance techniques such as uncoordinated checkpointing on reliability and energy consumption.

General perspective: energy consumption of the interconnect technology

Another interesting future direction should be the energy consumption of the interconnect technology. According to Biswas et al. [14], when systems grow 10 times, memory bandwidth needs to grow by at least 20 times so that applications can run efficiently. In this thesis, we have focused mainly on the energy consumption of the computational components of the machines. Indeed, the speed scaling technique is a technique where we reduce the CPU clock to minimize the energy consumption of the computation. However, the energy consumption of the interconnect technology is also one of the most critical barriers to Exascale [1]. For instance, it is known that if we follow standard JEDEC memory technology roadmap, the power consumption of a feasible Exascale system design (using 0.2 bytes/flop memory bandwidth balance) will be greater than 70 MW due to memory power consumption only [111]! This is not acceptable when Exascale systems are expected to have an energy consumption not exceeding 20 MW. Communications can occur at several levels:

- movements between processor cores and memory;
- movements between processors;
- movements between clusters.

A high-performance, energy-efficient interconnect is necessary to a future Exascale system. We need to assess the reliability of the interconnect in addition to that of computational resources.

Bibliography

[1] Advanced Scientific Computing Advisory Committee (ASCAC), "Ten technical approaches to address the challenges of Exascale computing," http://science.energy.gov/~/media/ascr/ascac/pdf/meetings/20140210/Top10reportFEB14.pdf.

[2] "AMD processors," http://www.amd.com.

[3] I. Assayad, A. Girault, and H. Kalla, "Tradeoff exploration between reliability power consumption and execution time," in *Proceedings of Computer Safety, Reliability and Security Conference (SAFECOMP)*. Washington, DC, USA: IEEE Computer Society Press, 2011.

[4] G. Aupy, "Source code and data for tri-criteria scheduling," http://gaupy.org/tri-criteria-scheduling. [Online]. Available: http://gaupy.org/tri-criteria-scheduling

[5] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation*. Springer Verlag, 1999.

[6] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society Press, 2003, pp. 113–121.

[7] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli, M. Peri, and S. Pezzini, "Fault-tolerant platforms for automotive safety-critical applications," in *Proc. of Int. Conf. on Compilers, Architectures and Synthesis for Embedded Systems*. ACM Press, 2003, pp. 170–177.

[8] N. Bansal, T. Kimbrel, and K. Pruhs, "Speed scaling to manage energy and temperature," *Journal of the ACM*, vol. 54, no. 1, pp. 1 – 39, 2007.

[9] E. Beigne, F. Clermidy, J. Durupt, H. Lhermet, S. Miermont, Y. Thonnart, T. Xuan, A. Valentian, D. Varreau, and P. Vivet, "An asynchronous power aware and adaptive NoC based circuit," in *Proceedings of the Symposium on VLSI Circuits*. IEEE Computer Society Press, June 2008, pp. 190–191.

[10] E. Beigne, F. Clermidy, S. Miermont, Y. Thonnart, A. Valentian, and P. Vivet, "A localized power control mixing hopping and super cut-off techniques within a GALS NoC," in *Proceedings of the International Conference on Integrated Circuit Design and Technology and Tutorial (ICICDT)*. IEEE Computer Society Press, June 2008, pp. 37–42.

[11] A. R. Benson, S. Schmit, and R. Schreiber, "Silent error detection in numerical time-stepping schemes," *CoRR*, vol. abs/1312.2674, 2013.

[12] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, vol. 15, 2008.

[13] V. Bharadwaj, T. G. Robertazzi, and D. Ghose, *Scheduling Divisible Loads in Parallel and Distributed Systems*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1996.

[14] R. Biswas, M. Aftosmis, C. Kiris, and B.-W. Shen, "Petascale computing: Impact on future nasa missions." Chapman and Hall/CRC Press, 2007, pp. 29–46.

[15] G. Bosilca, R. Delmas, J. Dongarra, and J. Langou, "Algorithm-based fault tolerance applied to high performance computing," *Journal of Parallel and Distributed Computing*, vol. 69, no. 4, pp. 410 –416, 2009.

[16] G. Bosilca, A. Bouteiller, E. Brunet, F. Cappello, J. Dongarra, A. Guermouche, T. Hérault, Y. Robert, F. Vivien, and D. Zaidouni, "Unified model for assessing checkpointing protocols at extreme-scale," *Concurrency and Computation: Practice and Experience*, October 2013.

[17] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien, "Checkpointing strategies for parallel jobs," in *Proceedings of the ACM/IEEE conference on SuperComputing (SC)*. ACM/IEEE Computer Society Press, 2011.

[18] M.-S. Bouguerra, T. Gautier, D. Trystram, and J.-M. Vincent, "A flexible checkpoint/restart model in distributed systems," in *International Conference on Parallel Processing and Applied Mathematics (PPAM)*, ser. LNCS, vol. 6067, 2010, pp. 206–215. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14390-8_22

[19] M.-S. Bouguerra, D. Trystram, and F. Wagner, "Complexity analysis of checkpoint scheduling with variable costs," *IEEE Transactions on Computers*, vol. 99, no. PrePrints, 2012.

[20] M. Bouguerra, A. Gainaru, L. Gomez, F. Cappello, S. Matsuoka, and N. Maruyama, "Improving the computing efficiency of HPC systems using a combination of proactive and preventive checkpointing," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society Press, May 2013, pp. 501–512.

[21] B. Bouteiller, P. Lemarinier, K. Krawezik, and F. Capello, "Coordinated checkpoint versus message log for fault tolerant MPI," in *Cluster Computing*. IEEE Computer Society Press, 2003, pp. 242–250.

[22] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[23] G. Bronevetsky and B. de Supinski, "Soft error vulnerability of iterative linear algebra methods," in *International Conference on Supercomputing (ICS)*. ACM Press, 2008, pp. 155–164.

[24] G. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo, *Soft Real-Time Systems: Predictability vs. Efficiency*. Springer series in Computer Science, 2005.

[25] F. Cappello, H. Casanova, and Y. Robert, "Preventive migration vs. preventive checkpointing for extreme scale supercomputers," *Parallel Processing Letters*, vol. 21, no. 2, pp. 111–132, 2011.

[26] F. Cappello, A. Geist, B. Gropp, L. V. Kalé, B. Kramer, and M. Snir, "Toward exascale resilience," *International Journal of High Performance Computing Applications*, vol. 23, no. 4, pp. 374–388, 2009.

[27] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, and W. P. Zeggert, "Proactive management of software aging," vol. 45, no. 2. Riverton, NJ, USA: IBM Corp., 2001, pp. 311–332.

[28] A. P. Chandrakasan and A. Sinha, "Jouletrack: A web based tool for software energy profiling," in *Design Automation Conference*. IEEE Computer Society Press, 2001, pp. 220–225.

[29] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," in *ACM Transactions on Computer Systems (TOCS)*, vol. 3(1). ACM Press, February 1985, pp. 63–75.

[30] G. Chen, K. Malkowski, M. Kandemir, and P. Raghavan, "Reducing power with performance constraints for parallel sparse applications," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society Press, April 2005, p. 8 pp.

[31] J.-J. Chen and T.-W. Kuo, "Multiprocessor energy-efficient scheduling for real-time tasks," in *Proceedings of International Conference on Parallel Processing (ICPP)*. IEEE Computer Society Press, 2005, pp. 13–20.

[32] J.-J. Chen and C.-F. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms," in *Proceedings of the International Workshop on Real-Time Computing Systems and Applications (RTCSA)*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2007, pp. 28–38.

[33] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J.-M. Vincent, and F. Wagner, "Random graph generation for scheduling simulations," in *Proceedings of the International Conference on Simulation Tools and Techniques (SIMUTools)*. ICST, 2010.

[34] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to algorithms," 2009.

[35] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," vol. 22, no. 3. Elsevier, 2006, pp. 303–312.

[36] V. Degalahal, L. Li, V. Narayanan, M. Kandemir, and M. J. Irwin, "Soft errors issues in low-power caches," *IEEE Transactions on Very Large Scale Integrated Systems*, vol. 13, pp. 1157–1166, October 2005.

[37] M. e. M. Diouri, O. Gluck, L. Lefèvre, and F. Cappello, "Energy considerations in checkpointing and fault tolerance protocols," in *Proceedings of the International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE Computer Society Press, 2012, pp. 1–6.

[38] M. e. M. Diouri, O. Gluck, L. Lefevre, and F. Cappello, "Ecofit: A framework to estimate energy consumption of fault tolerance protocols for HPC applications," in *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*. IEEE Computer Society Press, 2013, pp. 522–529.

[39] J. Dongarra, P. Beckman, P. Aerts, F. Cappello, T. Lippert, S. Matsuoka, P. Messina, T. Moore, R. Stevens, A. Trefethen, and M. Valero, "The international exascale software project: A call to cooperative action by the global high-performance community," *International Journal of High Performance Computing Applications*, vol. 23, no. 4, pp. 309–322, 2009.

[40] J. Dongarra, T. Hérault, and Y. Robert, "Revisiting the double checkpointing algorithm," in *Workshop on Advances in Parallel and Distributed Computational Models (APDCM)*. IEEE Computer Society Press, 2013.

[41] M. Drozdowski, "Divisible load," in *Scheduling for Parallel Processing*, ser. Computer Communications and Networks. Springer, 2009, pp. 301–365.

[42] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys*, vol. 34, no. 3, pp. 375–408, 2002.

[43] K. Ferreira, J. Stearley, J. H. I. Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold, "Evaluating the viability of process replication reliability for exascale systems," in *Proceedings of the ACM/IEEE conference on SuperComputing (SC)*. ACM/IEEE Computer Society Press, 2011.

[44] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell, "Detection and correction of silent data corruption for large-scale high-performance computing," in *Proceedings of the ACM/IEEE conference on SuperComputing (SC)*. ACM/IEEE Computer Society Press, 2012.

[45] E. W. Fulp, G. A. Fink, and J. N. Haack, "Predicting computer system failures using support vector machines," in *Proceedings of the First USENIX conference on Analysis of system logs*. USENIX Association, 2008.

[46] A. Gainaru, F. Cappello, and W. Kramer, "Taming of the shrew: Modeling the normal and faulty behavior of large-scale HPC systems," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society Press, 2012.

[47] A. Gainaru, F. Cappello, W. Kramer, and M. Snir, "Fault prediction under the microscope - a closer look into HPC systems," in *Proceedings of the ACM/IEEE conference on SuperComputing (SC)*. ACM/IEEE Computer Society Press, 2012.

[48] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[49] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters," in *Proceedings of the ACM/IEEE conference on SuperComputing (SC)*. ACM/IEEE Computer Society Press, 2005, p. 34.

[50] E. Gelenbe, "On the optimum checkpoint interval," *Journal of the ACM*, vol. 26, no. 2, pp. 259–270, 1979.

[51] E. Gelenbe and D. Derochette, "Performance of rollback recovery systems under intermittent failures," *Communications of the ACM*, vol. 21, no. 6, pp. 493–499, 1978.

[52] E. Gelenbe and M. Hernández, "Optimum checkpoints with age dependent failures," *Acta Informatica*, vol. 27, no. 6, pp. 519–531, 1990.

[53] A. Girault, E. Saule, and D. Trystram, "Reliability versus performance for critical applications," *J. Parallel Distrib. Comput.*, vol. 69, pp. 326–336, March 2009.

[54] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *Journal of Solid-State Circuits*, vol. 31, no. 9, pp. 1277 –1284, September 1996.

[55] ——, "Bounds on multiprocessing timing anomalies," *SIAM Journal on Applied Mathematics*, vol. 17, pp. 416–429, 1969.

[56] P. Grosse, Y. Durand, and P. Feautrier, "Methods for power optimization in SOC-based data flow systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 14, pp. 38:1–38:20, June 2009. [Online]. Available: http://doi.acm.org/10.1145/1529255.1529260

[57] A. Guermouche, T. Ropars, E. Brunet, M. Snir, and F. Cappello, "Uncoordinated checkpointing without domino effect for send-deterministic MPI applications," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society Press, 2011, pp. 989–1000.

[58] T. Heath, R. P. Martin, and T. D. Nguyen, "Improving cluster availability using workstation validation," *SIGMETRICS Perf. Eval. Rev.*, vol. 30, no. 1, 2002.

[59] E. Heien, D. Kondo, A. Gainaru, D. LaPine, B. Kramer, and F. Cappello, "Modeling and tolerating heterogeneous failures in large parallel systems," in *Proceedings of the ACM/IEEE conference on SuperComputing (SC)*. ACM/IEEE Computer Society Press, 2011.

[60] M. Heroux and M. Hoemmen, "Fault-tolerant iterative methods via selective reliability," Sandia National Laboratories, Research report SAND2011-3915 C, 2011.

[61] J. Hong, S. Kim, Y. Cho, H. Yeom, and T. Park, "On the choice of checkpoint interval using memory usage profile and adaptive time series analysis," in *Proceedings of the Pacific Rim Internation Symposium on Dependable Computing (PRDC)*. IEEE Computer Society Press, 2001.

[62] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi, "Profile-based optimization of power performance by using dynamic voltage scaling on a PC cluster," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society Press, 2006.

[63] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Transactions on Computers*, vol. 33, no. 6, pp. 518–528, 1984.

[64] A. A. Hwang, I. A. Stefanovici, and B. Schroeder, "Cosmic rays don't strike twice: Understanding the nature of dram errors and the implications for system design," *SIGARCH Comput. Archit. News*, vol. 40, no. 1, pp. 111–122, 2012.

[65] ——, "Voltage scheduling problem for dynamically variable voltage processors," in *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*. ACM Press, 1998, pp. 197–202.

[66] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proceedings of the Design Automation Conference (DAC)*. New York, NY, USA: ACM Press, 2004, pp. 275–280.

[67] H. Jin, Y. Chen, H. Zhu, and X.-H. Sun, "Optimizing HPC fault-tolerant environment: An analytical approach," in *Proceedings of International Conference on Parallel Processing (ICPP)*, 2010, pp. 525–534.

[68] H. Kawaguchi, G. Zhang, S. Lee, and T. Sakurai, "An LSI for VDD-Hopping and MPEG4 system based on the chip," in *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*. IEEE Computer Society Press, May 2001.

[69] O. Kella and W. Stadje, "Superposition of renewal processes and an application to multi-server queues," *Statistics & probability letters*, vol. 76, no. 17, pp. 1914–1924, 2006.

[70] K. H. Kim, R. Buyya, and J. Kim, "Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters," in *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*. IEEE Computer Society Press, May 2007, pp. 541–548.

[71] P. Kogge and J. Shalf, "Exascale computing trends: Adjusting to the "new normal" in computer architecture." IEEE Computer Society Press, 2013.

[72] N. Kolettis and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *International Symposium on Fault-Tolerant Computing (FTCS)*. IEEE Computer Society Press, 1995, p. 381.

[73] D. Kondo, B. Javadi, A. Iosup, and D. Epema, "The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems," *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, vol. 0, pp. 398–407, 2010.

[74] K. Lahiri, A. Raghunathan, S. Dey, and D. Panigrahi, "Battery-driven system design: A new frontier in low power design," in *Proceedings of the Asia South Pacific Design Automation Conference (ASPDAC)*. ACM Press, 2002, pp. 261 –267.

[75] P. Langen and B. Juurlink, "Leakage-aware multiprocessor scheduling," *J. Signal Process. Syst.*, vol. 57, no. 1, pp. 73–88, 2009.

[76] M. Larabel, "Intel EIST SpeedStep." [Online]. Available: http://www.phoronix.com/

[77] ——, "Run-time voltage hopping for low-power real-time systems," in *Proceedings of the Design Automation Conference (DAC)*. ACM Press, 2000, pp. 806–809.

[78] Y. Li, Z. Lan, P. Gujrati, and X. Sun, "Fault-aware runtime strategies for high-performance computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 4, pp. 460–473, 2009.

[79] Y. Liang, Y. Zhang, H. Xiong, and R. K. Sahoo, "Failure prediction in IBM BlueGene/L event logs," in *Proceedings of the International Conference on Data Mining series (ICDM)*. IEEE Computer Society Press, 2007, pp. 583–588.

[80] Y. Ling, J. Mi, and X. Lin, "A variational calculus approach to optimal checkpoint placement," *IEEE Transactions on Computers*, pp. 699–708, 2001.

[81] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S. Scott, "An optimal checkpoint/restart model for a large scale high performance computing system," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society Press, 2008.

[82] G. Lu, Z. Zheng, and A. A. Chien, "When is multi-version checkpointing needed," in *3rd Workshop for Fault-tolerance at Extreme Scale (FTXS)*. ACM Press, 2013.

[83] R. E. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM J. Res. Dev.*, vol. 6, no. 2, pp. 200–209, 1962.

[84] "Maple sheets for the experiments," http://graal.ens-lyon.fr/~yrobert/error-detection/.

[85] R. Melhem, D. Mosse, and E. Elnozahy, "The interplay of power management and fault recovery in real-time systems," *IEEE Transactions on Computers*, vol. 53, p. 2004, 2003.

[86] E. Meneses, O. Sarood, and L. V. Kalé, "Assessing Energy Efficiency of Fault Tolerance Protocols for HPC Systems," in *Proceedings of the International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. New York, USA: IEEE Computer Society Press, October 2012.

[87] S. Miermont, P. Vivet, and M. Renaudin, "A power supply selector for energy- and area-efficient local dynamic voltage scaling," in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*. Springer Berlin / Heidelberg, 2007, vol. 4644, pp. 556–565.

[88] M. P. Mills, "The internet begins with coal," *Environment and Climate News*, 1999.

[89] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[90] A. Moody, G. Bronevetsky, K. Mohror, and B. R. de Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *Proceedings of the ACM/IEEE conference on SuperComputing (SC)*. ACM/IEEE Computer Society Press, 2010, pp. 1–11.

[91] X. Ni, E. Meneses, and L. V. Kalé, "Hiding checkpoint overhead in HPC applications with a semi-blocking algorithm," in *Cluster Computing*. IEEE Computer Society Press, 2012.

[92] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York: Springer, 2006.

[93] T. Okuma, H. Yasuura, and T. Ishihara, "Software energy reduction techniques for variable-voltage processors," *IEEE Design & Test of Computers*, vol. 18, no. 2, pp. 31 –41, March 2001.

[94] R. Oldfield, S. Arunagiri, P. Teller, S. Seelam, M. Varela, R. Riesen, and P. Roth, "Modeling the impact of checkpoints on next-generation systems," in *Proceedings of the Conference on Mass Storage Systems and Technologies (MSST)*. IEEE Computer Society Press, 2007, pp. 30–46.

[95] R. A. Oldfield, S. Arunagiri, P. J. Teller, S. Seelam, M. R. Varela, R. Riesen, and P. C. Roth, "Modeling the impact of checkpoints on next-generation systems," in *Proceedings of the Conference on Mass Storage Systems and Technologies (MSST)*. IEEE Computer Society Press, 2007, pp. 30–46.

[96] A. J. Oliner, R. K. Sahoo, J. E. Moreira, M. Gupta, and A. Sivasubramaniam, "Fault-aware job scheduling for BlueGene/L systems," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society Press, 2004, pp. 64–73.

[97] T. Ozaki, T. Dohi, H. Okamura, and N. Kaio, "Distribution-free checkpoint placement algorithms based on min-max principle," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, pp. 130–140, 2006.

[98] J. S. Plank and M. G. Thomason, "Processor allocation and checkpoint interval selection in cluster computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, p. 1590, 2001.

[99] P. Pop, K. H. Poulsen, V. Izosimov, and P. Eles, "Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems," in *Proceedings of the International Conference on Hardware/software codesign and system synthesis (CODES+ISSS)*. ACM Press, 2007, pp. 233–238.

[100] R. B. Prathipati, "Energy efficient scheduling techniques for real-time embedded systems," Master's thesis, Texas A&M University, May 2004.

[101] K. Pruhs, R. van Stee, and P. Uthaisombut, "Speed scaling of tasks with precedence constraints," *Theory of Computing Systems*, vol. 43, pp. 67–80, 2008.

[102] R. Rajachandrasekar, A. Moody, K. Mohror, and D. K. D. Panda, "A 1 PB/s file system to checkpoint three million MPI tasks," in *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, ser. HPDC '13. New York, NY, USA: ACM, 2013, pp. 143–154.

[103] V. J. Rayward-Smith, F. W. Burton, and G. J. Janacek, "Scheduling parallel programs assuming preallocation," in *Scheduling Theory and its Applications*, P. Chrétienne, E. G. Coffman Jr., J. K. Lenstra, and Z. Liu, Eds. John Wiley and Sons, 1995.

[104] Y. Robert, F. Vivien, and D. Zaidouni, "On the complexity of scheduling checkpoints for computational workflows," in *Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS)*. IEEE Computer Society Press, 2012.

[105] S. M. Ross, *Introduction to Probability Models, Tenth Edition*. Academic Press, 2009.

[106] W. Rudin, *Principles of mathematical analysis*, 3rd ed. New York: McGraw-Hill Book Co., 1976, international Series in Pure and Applied Mathematics.

[107] P. Sao and R. Vuduc, "Self-stabilizing iterative solvers," in *Proc. Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ser. ScalA '13. ACM, 2013.

[108] V. Sarkar *et al.*, "Exascale software study: Software challenges in extreme scale systems," 2009, white paper available at: http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/ECSS%20report%20101909.pdf.

[109] K. Sato, A. Moody, K. Mohror, T. Gamblin, B. R. de Supinski, N. Maruyama, and S. Matsuoka, "Design and modeling of a non-blocking checkpointing system," in *Proceedings of the ACM/IEEE conference on SuperComputing (SC)*. ACM/IEEE Computer Society Press, 2012.

[110] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*. IEEE Computer Society Press, 2006, pp. 249–258.

[111] J. Shalf, S. Dosanjh, and J. Morrison, "Exascale computing technology challenges," in *Internation Conference on High Performance Computing for Computational Science (VECPAR)*, ser. LNCS 6449. Springer-Verlag, 2011, pp. 1–25.

[112] ——, "Fault tolerant preconditioned conjugate gradient for sparse linear system solution," in *International Conference on Supercomputing (ICS)*. ACM Press, 2012.

[113] S. M. Shatz and J.-P. Wang, "Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems," *IEEE Transactions on Reliability*, vol. 38, pp. 16–27, 1989.

[114] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Transactions on Architecture and Code Optimization*, vol. 1, no. 1, pp. 94–125, 2004.

[115] J. A. Stankovic, K. Ramamritham, and M. Spuri, *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 1998.

[116] S. Toueg and O. Babaoglu, "On the optimum checkpoint selection problem," *SIAM Journal Computing*, vol. 13, no. 3, pp. 630–649, 1984.

[117] L. Wang, P. Karthik, Z. Kalbarczyk, R. Iyer, L. Votta, C. Vick, and A. Wood, "Modeling coordinated checkpointing for large-scale supercomputers," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*. IEEE Computer Society Press, 2005, pp. 812–821.

[118] L. Wang, G. von Laszewski, J. Dayal, and F. Wang, "Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs," in *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*. IEEE Computer Society Press, May 2010, pp. 368 –377.

[119] Y.-M. Wang, P.-Y. Chung, I.-J. Lin, and W. K. Fuchs, "Checkpoint space reclamation for uncoordinated checkpointing in message-passing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 5, pp. 546–554, 1995.

[120] J. Wingstrom, "Overcoming The Difficulties Created by the Volatile Nature of Desktop Grids Through Understanding, Prediction and Redundancy," Ph.D. dissertation, University of Hawai'i at Manoa, 2009.

[121] ——, "Minimizing expected energy consumption in real-time systems through dynamic voltage scaling," *ACM Transactions on Computer Systems (TOCS)*, vol. 25, no. 4, p. 9, 2007.

[122] L. Yang and L. Man, "On-line and off-line DVS for fixed priority with preemption threshold scheduling," in *Proceedings of the International Conference on Embedded Software and Systems (ICESS)*, May 2009, pp. 273 –280.

[123] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. Washington, DC, USA: IEEE Computer Society Press, 1995, p. 374.

[124] J. W. Young, "A first order approximation to the optimum checkpoint interval," vol. 17, no. 9, 1974, pp. 530–531.

[125] L. Yu, Z. Zheng, Z. Lan, and S. Coghlan, "Practical online failure prediction for BlueGene/P: Period-based vs event-driven," in *Proceedings of the International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE Computer Society Press, 2011, pp. 259–264.

[126] Y. Zhang and K. Chakrabarty, "Energy-aware adaptive checkpointing in embedded real-time systems," in *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*. IEEE Computer Society Press, 2003, p. 10918.

[127] Y. Zhang, X. S. Hu, and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," in *Proceedings of the Design Automation Conference (DAC)*. New York, NY, USA: ACM Press, 2002, pp. 183–188.

[128] G. Zheng, X. Ni, and L. Kalé, "A scalable double in-memory checkpoint and restart scheme towards exascale," in *Proceedings of the International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE Computer Society Press, 2012.

[129] G. Zheng, L. Shi, and L. V. Kalé, "FTC-Charm++: An in-memory checkpoint-based fault tolerant runtime for Charm++ and MPI," in *Cluster Computing*. IEEE Computer Society Press, 2004.

[130] Z. Zheng and Z. Lan, "Reliability-aware scalability models for high performance computing," in *Cluster Computing*. IEEE Computer Society Press, 2009.

[131] Z. Zheng, Z. Lan, R. Gupta, S. Coghlan, and P. Beckman, "A practical failure prediction with location and lead time for BlueGene/P," in *Proceedings of the International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE Computer Society Press, 2010, pp. 15–22.

[132] D. Zhu, "Reliability-aware dynamic energy management in dependable embedded real-time systems," in *Real-Time and Embedded Technology and Applications Symposium*. IEEE Computer Society Press, 2006, pp. 397–407.

[133] D. Zhu and H. Aydin, "Energy management for real-time embedded systems with reliability requirements," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE Computer Society Press, 2006, pp. 528–534.

[134] D. Zhu, R. Melhem, and D. Mossé, "The effects of energy management on reliability in real-time embedded systems," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Washington, DC, USA: IEEE Computer Society Press, 2004, pp. 35–40.

Publications

Articles in international refereed journals

[J1] G. Aupy, A. Benoit, F. Dufossé, and Y. Robert, "Reclaiming the energy of a schedule: Models and algorithms," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 11, pp. 1505–1523, 2013.

[J2] G. Aupy, A. Benoit, J. Matthieu, and Y. Robert, "Power-aware replica placement in tree networks with multiple servers per client," in *Sustainable Computing: Informatics and Systems*. Elsevier, 2014.

[J3] G. Aupy and O. Bournez, "On the number of binary-minded individuals required to compute $\sqrt{\frac{1}{2}}$," *Theoretical Computer Science*, vol. 412, no. 22, pp. 2262–2267, 2011.

[J4] G. Aupy, Y. Robert, F. Vivien, and D. Zaidouni, "Checkpointing algorithms and fault prediction," *Journal of Parallel and Distributed Computing*, vol. 74, no. 2, pp. 2048–2064, 2014.

Articles in international refereed conferences

[C1] L. Atkins, G. Aupy, D. Cole, and K. Pruhs, "Speed scaling to manage temperature," in *Theory and Practice of Algorithms in (Computer) Systems (TAPAS)*. Springer-Verlag, 2011, pp. 9–20.

[C2] G. Aupy, A. Benoit, and Y. Robert, "Energy-aware scheduling under reliability and makespan constraint," in *Proceedings of the IEEE International Conference on High Performance Computing (HiPC)*. IEEE Computer Society Press, December 2012.

[C3] G. Aupy, A. Benoit, F. Dufossé, and Y. Robert, "Brief announcement: Reclaiming the energy of a schedule, models and algorithms," in *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM Press, 2011, pp. 135–136.

[C4] G. Aupy, A. Benoit, T. Hérault, Y. Robert, and J. Dongarra, "Optimal checkpointing period: Time vs. energy," in *Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, ser. LNCS. Springer-Verlag, 2013.

[C5] G. Aupy, A. Benoit, T. Hérault, Y. Robert, F. Vivien, and D. Zaidouni, "On the combination of silent error detection and checkpointing," in *Proceedings of the Pacific Rim Internation Symposium on Dependable Computing (PRDC)*. IEEE Computer Society Press, 2013.

[C6] G. Aupy, A. Benoit, J. Matthieu, and Y. Robert, "Power-aware replica placement in tree networks with multiple servers per client," in *Proceedings of Euro-Par: Parallel Processing*. Springer Verlag, 2014.

[C7] G. Aupy, A. Benoit, R. Melhem, P. Renaud-Goud, and Y. Robert, "Energy-aware checkpointing of divisible tasks with soft or hard deadlines," in *International Green Computing Conference (IGCC)*. IEEE Computer Society Press, 2013, pp. 1–8.

[C8] G. Aupy, M. Faverge, Y. Robert, J. Kurzak, P. Luszczek, and J. Dongarra, "Implementing a systolic algorithm for QR factorization on multicore clusters with PaRSEC," in *Workshop on Productivity and Performance (PROPER)*, ser. LNCS 8374. Springer-Verlag, 2013.

[C9] G. Aupy, Y. Robert, F. Vivien, and D. Zaidouni, "Checkpointing strategies with prediction windows," in *Proceedings of the Pacific Rim Internation Symposium on Dependable Computing (PRDC)*. IEEE Computer Society Press, 2013.

Research reports

[RR1] G. Aupy and A. Benoit, "Approximation algorithms for energy, reliability and makespan optimization problems," INRIA, Tech. Rep. RR-8107, October 2012. [Online]. Available: http://hal.inria.fr/hal-00742754

[RR2] G. Aupy, A. Benoit, F. Dufossé, and Y. Robert, "Reclaiming the energy of a schedule: Models and algorithms," INRIA, Tech. Rep. RR-7598, April 2011. [Online]. Available: http://hal.inria.fr/inria-00584944

[RR3] G. Aupy, A. Benoit, J. Matthieu, and Y. Robert, "Power-aware replica placement in tree networks with multiple servers per client," INRIA, Rapport de recherche RR-8474, February 2014. [Online]. Available: http://hal.inria.fr/hal-00949252

[RR4] G. Aupy, A. Benoit, R. Melhem, P. Renaud-Goud, and Y. Robert, "Energy-aware checkpointing of divisible tasks with soft or hard deadlines," INRIA, Tech. Rep. RR-8238, February 2013. [Online]. Available: http://hal.inria.fr/hal-00788641

[RR5] G. Aupy, A. Benoit, and Y. Robert, "Energy-aware scheduling under reliability and makespan constraints," INRIA, Tech. Rep. RR-7757, February 2012. [Online]. Available: http://hal.inria.fr/inria-00630721

[RR6] G. Aupy, Y. Robert, F. Vivien, and D. Zaidouni, "Impact of fault prediction on checkpointing strategies," INRIA, Tech. Rep. RR-8023, October 2012, this report is rendered obsolete by RR-8237 and RR-8239 which cover the integrality of this report in a more precise fashion. [Online]. Available: http://hal.inria.fr/hal-00720401

[RR7] ——, "Checkpointing algorithms and fault prediction," INRIA, Tech. Rep. RR-8237, February 2013. [Online]. Available: http://hal.inria.fr/hal-00788313

[RR8] ——, "Checkpointing strategies with prediction windows," INRIA, Tech. Rep. RR-8239, February 2013. [Online]. Available: http://hal.inria.fr/hal-00789109

[RR9] ——, "Comments on "Improving the computing efficiency of HPC systems using a combination of proactive and preventive checkpoint"," INRIA, Rapport de recherche RR-8318, June 2013. [Online]. Available: http://hal.inria.fr/hal-00836629

[RR10] G. Aupy, M. Shantharam, A. Benoit, Y. Robert, and P. Raghavan, "Co-Scheduling Algorithms for High-Throughput Workload Execution," INRIA, Tech. Rep. RR-8293, April 2013. [Online]. Available: http://hal.inria.fr/hal-00819036

[RR11] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, and M. Snir, “Scheduling the I/O of HPC applications under congestion,” INRIA, Rapport de recherche RR-8519, April 2014. [Online]. Available: http://hal.inria.fr/hal-00983789